Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

UTVECKLING AV DELSYSTEM I EN WEBBAPPLIKATION

-räddningskampanjer för prenumerationer

Johannes Sundqvist

2022:28

Datum för godkännande: 18.05.2022
Handledare:Björn-Erik Zetterman

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Utbildningsprogrammet för Informationsteknik
Författare:	Johannes Sundqvist
Arbetets namn:	Utveckling av delsystem i en webbapplikation - räddningskampanjer för prenumerationer
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	N/A

Abstrakt

I mitt examensarbete skriver jag om grunder i webbsidor/webbsideutveckling samt *Payway* Räddningskampanjer. Examensarbetet fick sin grund inom mitt yrke, där jag under sommaren/hösten 2021 var med och utvecklade ett delsystem till ett stort och befintligt system. Utvecklingen av detta delsystem har gjorts via ett flertal, huvudsakligen objektorienterade programmeringsspråk. Systemet kom inte i bruk före senare delen av hösten 2021, vilket betyder att det finns ypperligt lite information att visualisera i form av grafer och tabeller. Detta arbete kommer ta förväntningar i beaktande, och den data som är visualiserad är endast uträkningar/förväntningar och inte egentliga siffror.

Nyckelord (sökord)

payway, räddnings kampanjer, webbsidor, webbutveckling, webbapplikationer, C#, ruby, prenumeration, prenumerationer

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2022:28	1458-1531	Svenska	34 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
27.04.2022	13.05.2022	18.05.2022

DEGREE THESIS
Åland University of Applied Sciences

Degree Programme:	Information Technology 2017
Author:	Johannes Sundqvist
Title:	Development of a partial system in a web application - retention campaigns for a subscription system
Academic Supervisor:	Björn-Erik Zetterman
Commissioned by:	N/A

Abstract

The topic of my degree thesis is Website development basics and Payway retention/save campaigns. The thesis got its foundation within my own profession and workplace, where I during the summer and autumn of 2021 worked with a team to develop a medium-sized system, to expand upon an already large system. The development of this subsystem has been done through several programming languages, though the majority of them are object-oriented languages.

The subsystem was put into production during late-autumn of 2021, which means there will be very little actual user-information to go on. All tables and graphs mentioning costs/revenue are calculated and simulated, and will as such only reflect and not be equal to the actual numbers.

Keywords

payway, retention campaigns, save campaigns, websites, web development, web,applications, c#, ruby, subscription, subscriptions

Serial number:	ISSN:	Language:	Number of pages:
2022:28	1458-1531	Swedish	34 pages

Handed in:	Date of presentation:	Approved:
27.04.2022	13.05.2022	18.05.2022

INNEHÅLLSFÖRTECKNING

1. INLEDNING

I detta examensarbete diskuteras grunderna i webbapplikationer samt hur räddningskampanjer har hjälpt prenumerationssystemet Payway[1]. Utvecklingen har skett med hjälp av Tulo-teamet[2] där jag personligen varit involverad, som är anställd under Adeprimo Ab[3]. Den primära utvecklingstiden var sommaren-hösten 2021, och pågår som kontinuerlig utveckling.

1.1 Syfte

Syftet med "Räddningskampanjerna" var att försöka få en kund, vars prenumeration till en digital eller fysisk tidning håller på gå ut, att stanna kvar, genom att erbjuda ett bättre erbjudande än vad som vanligtvis skulle vara tillgängligt. Detta födde konceptet med en s.k. "räddningskampanjtrappa", där kundflödet ska gå från en ordinarie prenumeration, till en "räddningskampanj", och vid ett avslutande inom samma "serie", ska nästa steg i denna trappa erbjudas (i detta fall, med det näst-bästa alternativet).

1.2 Metod

Metoden som användes för att implementera detta koncept var, som tidigare nämnt, ett "trapp"-koncept. Genomgången av denna trappa är endast nedåtgående, vilket betyder att en kund ska inte kunna gå från Räddningskampanj 2 till Räddningskampanj 1, men snarare andra vägen. Kunden ska inte heller kunna "hoppa" något steg, men hamnar alltid till steg 2, ifall denna befinner sig på steg 1 just nu.

Från ett mera tekniskt perspektiv implementerades majoriteten av denna logik i programmeringsspråket Ruby[4]. I samband med detta har även kodens funktionalitet testats via det s.k. RSpec[5]-biblioteket. Detta bibliotek lägger grunderna för "Behaviour driven development" och tillåter testning av kodens funktionalitet, genom olika skräddarsydda scenarion. Till exempel kunde detta innebära testscenarion som skulle uppstå när en användare matat in helt korrekt information, men även motsatsen - det vill säga, när en

[1] https://worldoftulo.com/payway/
[2] https://worldoftulo.com/
[3] https://www.adeprimo.se/
[4] https://www.ruby-lang.org/en/
[5] https://rspec.info/

användare gjort något fel, och man enligt programlogiken förväntar sig ett logiskt och hjälpsamt felmeddelande. Mera om RSpec och backendtestning under kapitel 2.

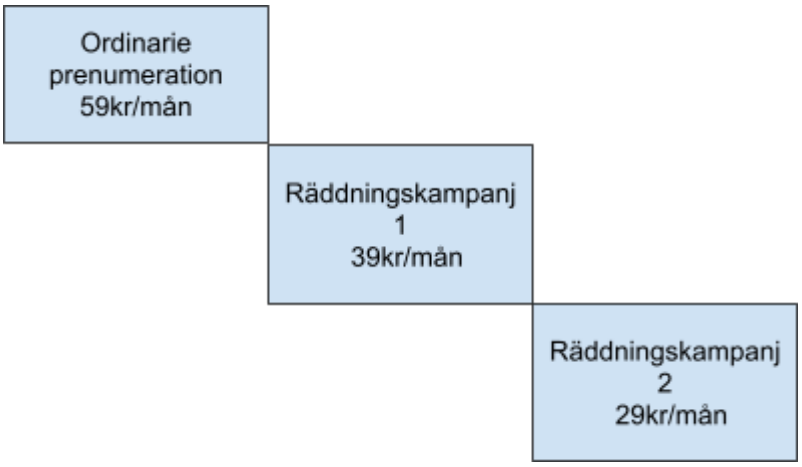
1.3 Avgränsningar

Detta koncept försökte implementeras som en ganska "vattentät" lösning, men några avgränsningar definierades:

1. Ifall en räddningskampanjtrappa både skapats och tagits bort inom perioden av att en kund har köpt en ordinarie prenumeration, så erbjuds ingen räddningskampanj.
2. Ifall en räddningskampanjtrappa uppdaterats så att flera eller färre steg kommit till, så är det detta som gäller. Inga "virtuella" trappor skapas för att en kund ska få den räddningskampanjtrappa som råkade vara aktiv precis under det tillfället som ordinarie prenumerationen påbörjades.
3. Räddningskampanjerna ingående i en räddningskampanjtrappa kan inte köpas utanför trappan.

1.4 Bakgrund

Bakgrunden till räddningskampanjerna var affärsmässigt enkla - ett sätt att få befintliga kunder att stanna kvar. För att få befintliga kunder att stanna kvar etablerades konceptet med "trappor" för att erbjuda kunden en fortsatt köpt tjänst. Vi hänvisar till Figur 1.



Figur 1: Enkel visualisering över hur en räddningskampanjtrappa i Payway kan se ut

De flesta som är vana med det grundläggande inom "business" vet att det är betydligt billigare att hålla kvar befintliga kunder, än att ta in nya kunder. I själva verket är det ungefär

5 gånger dyrare (Landis, n.d.) att försöka skaffa nya kunder än hålla kvar befintliga, och enligt forskning är chansen att lyckas sälja något till befintliga kunder mellan 60-70%, medan den procenten ligger på 5-20% för nya kunder (Landis, n.d.).

I ren konkrethet betyder detta att man, med en fiktiv kundbas på 10.000 pers, skulle lyckas sälja till allt mellan 500 och 2000 personer ifall varje person skulle vara "unik" och inte utföra vidare köp, medan man med kunder som är nöjda och fortsätter med tjänsten skulle kunna sälja till allt mellan 6000 och 7000 personer.

Genomgången av denna trappa är endast nedåtgående, vilket betyder att en kund ska inte kunna gå från Räddningskampanj 2 till Räddningskampanj 1, men snarare andra vägen. Kunden ska inte heller kunna "hoppa" något steg, men hamnar alltid till steg 2, ifall denna befinner sig på steg 1 just nu.

Vi hänvisar till kapitel 5 och framåt för vidare diskussion på hur räddningskampanjer fungerar, och fungerat i systemet Payway.

1.5 Definitioner / ordlista

- **Frontend:** Visuella delen av en webbsida, något som slutkund/slutanvändare ser.
- **Backend:** Logik-hanteringen för en webbsida, något som slutkund/slutanvändare inte ser, men blir direkt berörd av.
- **Gem(s):** Paket inom Ruby som används för att ha tillgång till olika funktioner på kodnivå. Kallas även paket, dependencies, imports.
- **IDE:** Integrated Development Environment. Textredigerare med extra funktioner för att underlätta programmering.
- **Database:** En kollektion av tabeller för att spara ner och lagra olika typer av data/information.
- **MVC:** Model-View-Controller
- **PAC:** Presentation-Abstraction-Control
- **API:** Application Programming Interface: ett interface som applikationer kan använda för att göra olika operationer, ofta mot en befintlig databas.

- **JSON:** Javascript Object Notation: ett sätt att skicka komplex data från t.ex. ett inmatningsfält till ett API utan att skapa problem i själva anropet.

2. TEORI

2.1 Språk

Under utvecklingen av Payway Räddningskampanjer användes huvudsakligen C# med ramverksversion .NET Core 3.1[6] för att stödja frontend, samt Ruby för att stödja backend. För att upprätta en databas används och användes MongoDb och MongoId.

2.1.1 Frontend

För att få räddningskampanjtrapporna konfigurerbara samt visualiserade mot slutkunderna behövs en frontend, och i detta fall valdes C# med ramverksversion .NET Core 3.1. För själva webbläsardokumentet användes HTML5, CSS3 och JavaScript inklusive det något kändare biblioteket jQuery[7].

```
@Model MyNamespace.MyViewModel
<html>
    <head>
        <title>My document</title>
        <link href=”style.css” rel=”stylesheet” />
    </head>
    <body>
        <div class=”wrapper”>
            @if(!string.IsNullOrEmpty(Model.Id)) {
               <!-- do something if we have an ID -->
            } else {
               <!-- otherwise do something else -->
            }
        </div>
    </body>
</html>
```

Figur 2.1: Illustrering över hur ett CSHTML dokument kan se ut. Skillnaden mellan ett HTML och CSHTML dokument är att CSHTML även stöder C# kod inom dokumentet.

[6] Ramverk för att kunna utveckla webbaserade applikationer inom C#. Läs mera här.
[7] Bibliotek för att underlätta anrop/utveckling av diverse JavaScript-baserade skript. Läs mera här.

```
using System;

namespace MyNamespace {
    public class MyViewModel {
        public string Id { get; set; }

        public MyViewModel(string id) {
            Id = id;
        }
    }
}
```

Figur 2.2: Illustrering över hur en vymodell i en MVC-arkitektur kan se ut.

```
using System;

public class MyController : Controller {
    private readonly IDataService _dataService;

    public MyController(IDataService dataService) {
        _dataService = dataService;
    }

    [HttpGet]
    public IActionResult Index() {
        return View(new MyViewModel(_dataService.GetId()));
    }
}
```

Figur 2.3: Illustrering över hur en controller kan se ut i en MVC-arkitektur. Här används även ett s.k. interface för att påbörja implementeringen av en klass för databas relaterade anrop.

2.1.2 HTML4 vs HTML5

Förutom att HTML blivit använt för att få uppmärkt den generella strukturen för en webbsida, finns det en version skillnad som kan lönas att hålla i åtanke här. Olikt andra versioner går det att kombinera HTML4 och HTML5 ifall man vill, och skilda uppdateringar/installationer behöver inte heller göras för att växla mellan de två versionerna. Vi hänvisar till tabell 1 för att få en idé över de generella skillnaderna mellan HTML4 och HTML5.

Tabell 1: Tabell över de huvudsakliga skillnaderna mellan HTML4 och HTML5. Notera att de ändringar som tabellen tar upp INTE är komplett, och andra ändringar fortfarande finns. (Difference between HTML and HTML5, 2018)

Funktion/element	**HTML4 Standard**	**HTML5 Standard**
Video & audio	Implementering av en Flash-player	<audio> och <video>-HTML taggar
JavaScript	Ej tillåtet	Tillåter skript att köra i bakgrunden
Drag- and drop	Ej tillåtet	Tillåtet
Mobilversioner	Fungerar, men ej optimalt	Fungerar på ett betydligt bättre sätt
Syntax	Taggar avslutas med >	Taggar avslutas med />

2.1.3 Alternativ till vald frontend

Enligt en webbsidas natur fanns och finns inte så många alternativ då det kommer till strukturen av en webbsida, dvs. HTML/CSS/JavaScript. Alternativen vi i detta skede kunde fundera på och möjligen välja mellan ligger ett steg “djupare”, där vårt val blev C#. Huvudsakligen valdes C# eftersom de s.k. Räddnningskampanjerna blev en modul i ett befintligt system, och att börja blanda med andra programmeringsspråk tillsammans med C# snabbt kan bli konstigt (eller rent av inte alls vara kompatibelt). Tar vi ett steg bak och läser av helheten betyder detta att Payway, dvs. hela systemet som ligger bakom, skulle behövt bli utvecklat inom ett annat språk. Här kan vi konstatera några möjligheter, alla med sina egna egenskaper (inklusive, men inte begränsade till):

- PHP
- Java
- Python
- .. med flera

Anledningen just C# valdes som frontend till Payway var inte allt för invecklat i sig - det var helt enkelt ett språk som fungerade bra för ändamålet, och de flesta utvecklare kände sig bekväma i språket. Från ett abstrakt perspektiv kan vi däremot ställa upp de huvudsakliga för-

och nackdelarna med de olika alternativen, för att få en idé om vad som "kunde varit". Vi hänvisar till tabell 2. Notera att detta är information som skrivs i dagsläget, och olika för- och nackdelar kan ha tagit annan form än hur det såg ut när Payway fick sin början.

Tabell 2: Tabell över för- och nackdelarna mellan C#, PHP, Java och Python. (Deremuk, 2021) (Advantages and Disadvantages of PHP, 2020) (Advantages and Disadvantages of PHP, 2020; Arora et al., n.d.)

Språk	**Fördelar**	**Nackdelar**
C#	OOP, MVC-mönster, Enhetstestning	.NET-baserat, behöver köras på en Windows-maskin
PHP	Enkelt att hitta webbhotell, att fånga upp form- eller URL data och att sätta upp från början	Fokuserat på mindre applikationer, både läsbarhet och prestanda lider av större applikationer
Java	Välutvecklad dokumentation, stort utval av tredjepartsbibliotek	Större krav på hårdvaru resurserna
Python	Open source, mycket flexibel på grund av PyPI-modulen	Snabbare utveckling, långsammare prestanda under körning

2.1.1.1 Arkitekturmodell

Inom frontend-miljön finns det även ett antal arkitekturmodeller som kan följas. För att få en idé över de vanligare arkitekturmodellerna, samt de huvudsakliga för- och nackdelarna per modell hänvisar vi till nedanstående tabell. Förkortningar förklaras under kapitel 1.4 Definitioner / Ordlista.

Tabell 3: Tabell som representerar ett antal olika arkitekturmodeller, samt deras huvudsakliga för- och nackdelar (Ferrara, 2014)

Modell	**Fördelar**	**Nackdelar**
MVC	- Möjlighet att binda flera vyer till samma datamodell - "Plug and play" koncept mellan view och controller	- Komplex - View och Controller är väldigt bundna
PAC	- "Plug and play" koncept med hjälp av modellens s.k. agents	- Komplex - Svårt att veta vad som är rätt mängd "agents"

Data-centered	- Erbjuder ett högt integritetsvärde - Mindre overhead	- Risk för duplikat på data nivå - Svårt att ändra på data modeller utan att beröra slutanvändare avsevärt
Blackboard	- Stöder experimentering av hypoteser - Enkelt att skala upp/ner	- Potentiella synkroniseringsproblem - Svårt att testa system som kör med modellen

Utöver detta kan vi också få en idé över hur olika modeller fungerar i verkligheten, och inte endast i teorin. För att få en idé över detta hänvisar vi till tabell 4. Förkortningar förklaras under kapitel 1.4 Definitioner / Ordlista.

Tabell 4: Tabell över vilka företag/organisationer som använder olika former av arkitekturmodeller. (Danylko, 2017) (Do You Know Any Examples of a PAC Design Pattern?, n.d.; Ferrara, 2014) (Data-Centered Architecture, n.d.) (Blackboard System, n.d.)

Modell	**Exempel**
MVC	Microsoft, StackOverflow, DELL
PAC	Drupal
Data-centered	CORBA, större biblioteksystem
Blackboard	Igenkänning- och identifikationssystem

2.1.2 Backend

För att förstå sig på hur den grundläggande backend-logiken fungerat, behöver vi först få en ungefärlig idé om just vad en backend *är*. En webbapplikation kan vi, i väldigt grova drag, dela upp i två delar: frontend, och backend. Frontend är det visuella som vi presenterar till användaren, hantering av saker användaren gör, utvisning av felmeddelanden osv. Vi kan lite se på det som applikationens “ansikte utåt”. Backend av en applikation är exakta motsatsen, och är det vi aldrig *visar* åt användaren, men det som tar hand om allt användaren gör i det stora hela. Ifall en användare registrerar sig kommer ett anrop gå från frontend, till backenden, för att därefter hanteras och sparas i en databas. Ifall en användare väljer att skapa ett inlägg, skickar vi ett anrop från frontend till backenden, gör förmodligen en och annan

kontroll för att se till att detta är något användaren får göra, och sparar därefter inlägget i en databas för andra att se. Utan en backend skulle största delen av applikationer i dagens läge helt enkelt inte fungera, medan utan en frontend skulle vi inte ha något att visa, även om det skulle finnas underliggande stöd för vad som helst.

Nu för att komma tillbaka till räddningskampanjer: För att kunna spara ner och erbjuda räddningskampanjtrapporna behövdes både logik för att räkna ut vilket steg i trappan en användare ligger på, samt för att få en databas representation/databas entitet. I detta fall används och användes Ruby, med ett tiotal gems, både skräddarsydda för eget bruk och allmänt distribuerade.

```ruby
module MyNamespace
    class MyClass
        attr_reader :id
        def initialize(id)
           @id = id
        end

        def get_id
           self.id
        end
    end
end
```

Figur 3: Illustrering över hur syntaxen för ett Ruby-projekt kan se ut.

2.1.3 Backend - exponering utåt[8]

Utöver detta behöver även våran backend exponera ett s.k. API. Detta för att kunna utföra anrop och skapa nya, spara befintliga, ta bort eller hämta ut entiteter. För att åstadkomma detta skapar vi s.k. *endpoints* som svarar med ett JSON-resultat, genom en av följande metoder:

[8] Mera om Payway's officiella API kan läsas på deras officiella dokumentation här.

GET

Används för att hämta ut data. Ofta då man besöker en webbplats via en webbläsare görs ett antal GET-anrop utan att man kanske tänker på det.

POST

Används för att skapa ny data. Ofta används denna ifall vi inte vill exponera vad som skickas in/ut, t.ex. vid inloggning eller registrering där man kommer åt sensitiv data.

PUT

Fungerar mycket likt ett POST-anrop, men används för uppdatering av befintlig data.

DELETE

Fungerar mycket likt ett POST-anrop, men används för borttagning av befintlig data.

2.1.4 Backend - testning

Utöver tillagd logik och API-exponering har även varje funktion i backend sina egna tester. Detta betyder att man genom ett enkelt kommando kan säkerställa funktionaliteten av varje modul/del för att se till att inget gått sönder under utvecklingens gång.

Dessa tester görs av Ruby's inbyggda RSpec[9], och baserar sig på att säkerställa funktionaliteten och pålitligheten av en specificerad komponent, snarare än prestandan. Uppbyggnaden av testerna baserar sig på att simulera verkligheten så långt det går och kontrollera ifall värdet som funktionen (eller API:erna) returnerar är förväntat värde. Är värdet inte det förväntade värdet markerar vi testet som fallerat, och går vidare till nästa test.

[9] Modul för att driva enhetstester i programmeringsspråket Ruby. Läs mera här.

```
# Simulate a test-suite for a GET endpoint accessible via /get_all
describe '#get_all_entities' do
 let!(:test_entity) { FactoryBot.create(:my_entity) }

 def do_get_all_entities
    get ‘/get_all’
 end

 it ‘returns all entities’ do
     cmd = JSON.parse(do_get_all_entities.body)
     result = cmd[“result”].first
     expect(result[“id”]).to eq test_entity.id
     expect(result[“name”]).to eq test_entity.name
 end
end
```

Figur 4: Exempel på hur ett testfall kan se ut i Ruby's RSpec. Notera att koden ovan endast är själva testet och all kringliggande kod som krävs för att utföra ett test inte finns med.

2.1.5 Alternativ till vald backend

Precis som vi kunde konstatera i kapitlet om frontend, finns det alternativ till backend-lösningen, som i detta fall blev Ruby. Problemet är att Räddningskampanjer återigen är en del av ett befintligt och stort system, vilket betyder att börja blanda med andra språk skulle betyda onödigt arbete och onödiga utmaningar. Med det sagt, så kan vi bege oss in på tanken vad som skulle hänt ifall något som inte är Ruby skulle blivit valt från första början. Några exempel inkluderar, men är inte begränsade till:

- Laravel
- Symfony
- ASP.NET MVC

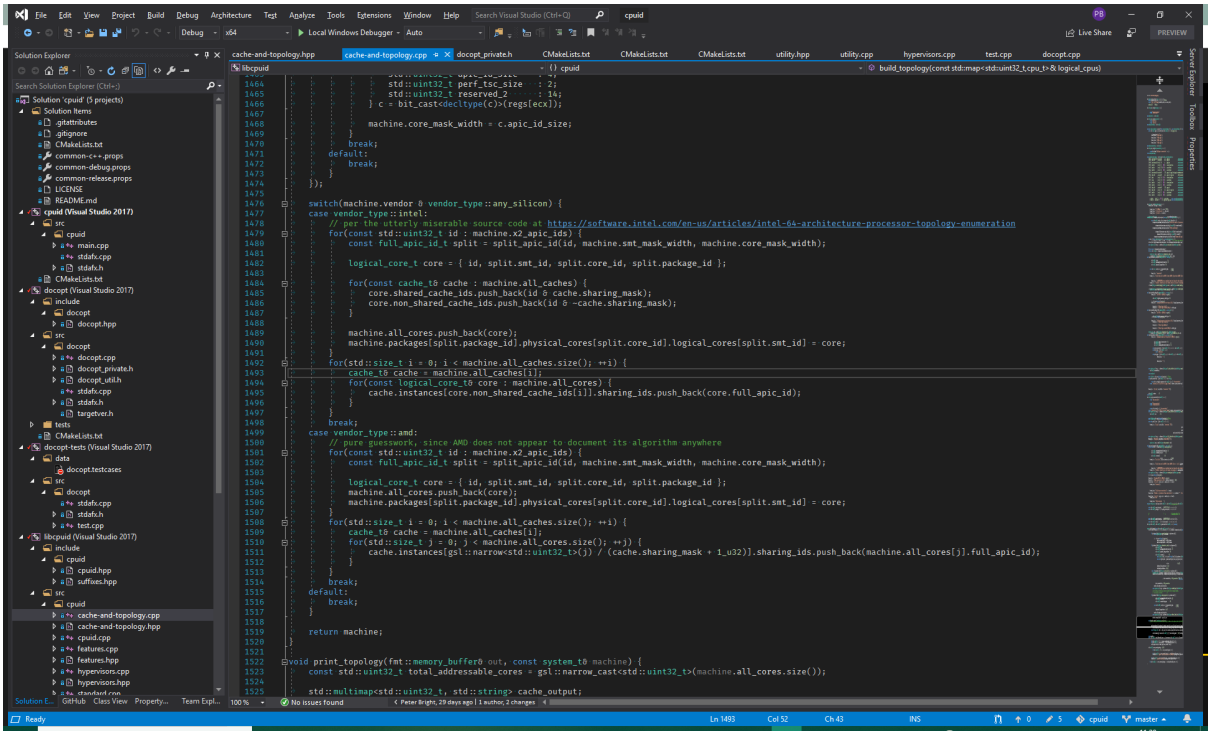
Anledningen Ruby valdes för att användas inom Payway är relativt enkel - under sin tid fanns det många *open source*-bibliotek som gjorde att utvecklingen framskred snabbt, och del av utvecklarna var även bekant/bekväma med språket sedan tidigare. Hur som helst kan vi få en abstrakt idé över vad som “kunde varit” genom att jämföra några för- och nackdelar med Ruby och de listade alternativen, hänvisande till tabell 5.

Tabell 5: Kompakt jämförelse mellan för- och nackdelarna mellan Ruby, Laravel, Symfony och ASP.NET MVC. (Omelia, 2020) (House, 2020; Omelia, 2020)

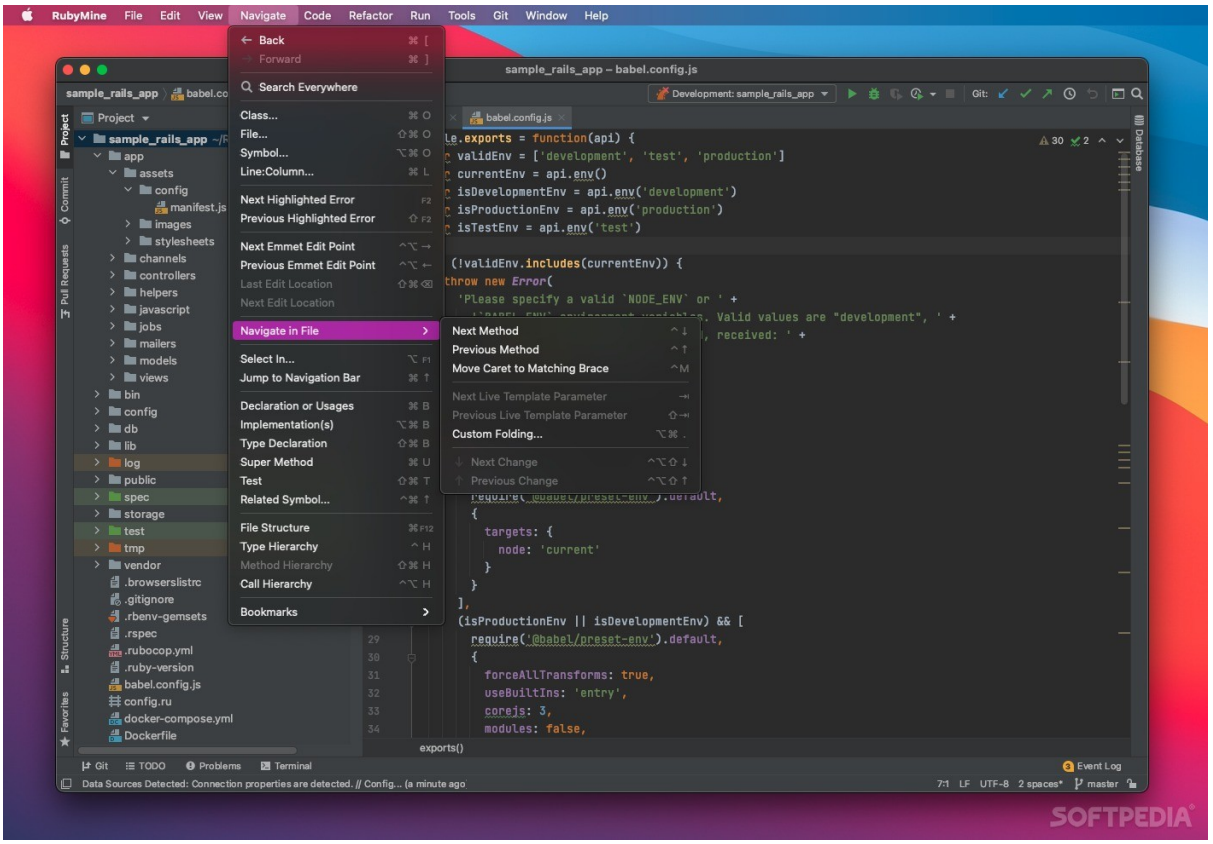
Språk	**Fördelar**	**Nackdelar**
Ruby	Välutvecklade tredjepartsbibliotek, eller s.k. “gems”	Potentiella prestanda-relaterade problem
Laravel	Bättre scalability, design likt MVC	Dålig support, både från funktions- och forumperspektiv
Symfony	Enkel att underhålla och testa, snabb utvecklingstid och hjälpsamma forum.	Potentiella prestanda-relaterade problem, kan vara utmanande att “komma igång”
ASP.NET MVC	Stabil och välstrukturerad i större applikationer. Utmärkt support från MS.	Komplex syntax, kräver en Windows-maskin för körning

2.2 Verktyg

Under utvecklingens gång användes Visual Studio 2019 CE och RubyMine som IDEs för frontend respektive backend (**I**ntegrated **D**evelopment **E**nvironment). För att övervaka kringliggande tjänster och appar användes Azure Portal, och för intern team-kommunikation användes Slack, Microsoft Teams, samt Jira.



Figur 5: Skärmdump från ett Visual Studio projekt, och hur detta kan se ut. Notera exponerad kod inte är del av Payway eller något underliggande/känt system. (Sudo Null Company, n.d.)



Figur 6: Skärmdump från ett RubyMine projekt, och hur detta kan se ut. Notera exponerad kod inte är del av Payway eller något underliggande/känt system. (Gatlan, n.d.)

2.3 Versionshantering

Under utvecklingens gång användes Bitbucket, som är ett versionshanteringsverktyg mycket likt Github. Kommandostrukturen ser exakt likadan ut, och principen är exakt densamma. Det som skiljer sig till Github är huvudsakligen användningsområdet. Här skiljer sig Bitbucket och Github genom att Bitbucket har bättre stöd för privata repositories jämfört med Github, och exponerar därav inte varken kod eller commits till allmänheten. (*Difference Between Bitbucket and GitHub*, 2020)

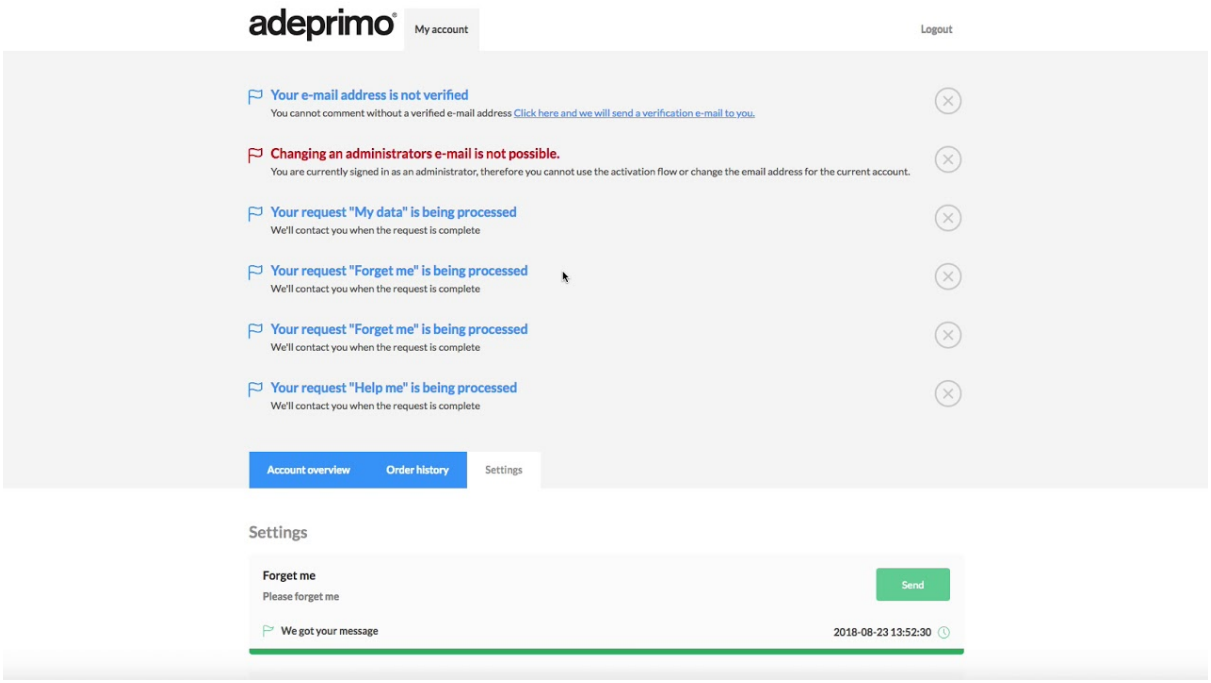
3. PRODUKTEN

Payway är i sin grund en existerande betallösning för fysiska samt digitala tidningar, härstammande i Sverige men som även börjar ta sig in på den Finländska marknaden. Systemet erbjuder en stor mängd funktionalitet, inklusive men inte begränsade till följande (*PayWay*, 2019):

- Skapa upp tidningspaket och kampanjer, med individuella betalperioder och regler
- Distribution av fysiska tidningar
- Integrera med (för branschen) kända integrationssystem såsom Adyen[10], Kayak[11], Klarna[12], och InfoSoft[13].
- Skapa upp en användar hierarki för redaktionens personal, med olika rättigheter och åtkomstnivåer.
- Administrera och hantera slutkundernas användarkonton, t.ex. för att sätta igång en prenumeration utan att slutkund själv ska behöva ta åtgärder, eller stänga av användare som missbrukar systemet.

Payway har även kringliggande produkter såsom BI-system, rapportsystem, med mera. Produkterna har sedan 2009 utvecklats med hjälp av Tulo-teamet[14] och ett 40-tal anställda, som är anställd under Adeprimo[15].

[10] Betalsystem för att kunna erhålla säkra köp inom sina egna system/plattformar. Läs mera här.
[11] System för att underhålla prenumerationer samt prenumeranter. Läs mera här.
[12] Betalsystem för att kunna erhålla säkra köp inom sina egna system/plattformar. Läs mera här.
[13] System för att underhålla prenumerationer samt prenumeranter. Läs mera här.
[14] https://worldoftulo.com/
[15] https://www.adeprimo.se/



Figur 7: Skärmdump över hur portalen kan se ut för en slutanvändare efter inloggning.

3.1 Flödet som kund i ett Payway baserat system

För att få en idé över hur flödet fungerar för en slutkund i Payway citerar vi följande från deras egna webbsida:

> "The flow in Tulo PayWay means that a customer creates an account and buys something, usually a subscription. The platform manages the account, transactions and the life cycle of the subscription. There is an "out of the box" solution with a purchase flow that is automatically created based on an offer and a customer portal. Here, end users can manage their cases, change tasks, manage their subscriptions etc. If you want to control the whole experience for the end customer, there is a powerful API to control the design and build your own business rules on the platform's APIs." (*PayWay*, 2019)

Flödet som kund (och slutkund) kan vi alltså se fungerar som det är. Det är relativt enkelt att börja använda i Payway's "enklaste" form, och även om möjligheten finns att bygga en egen erfarenhet åt slutkunderna, så är detta inget obligatoriskt.

> "Tulo PayWay has a broad integration layer for flexible integration with the systems needed in an ecosystem. Today there are integrations with solutions for CMSs such as Infomaker,

> WordPress, Drupal, Polopoly, Escenic and more. The integrations are based on the open standard OAuth2 for authentication and authorization.” (*PayWay*, 2019)

Förutom det huvudsakliga flödet i Payway så kan vi även se att integrationer mot välkända och “tried and tested” system finns till en godtycklig utsträckning. Ifall nuvarande sidor och tjänster erbjuds via någon befintligt CMS (Content Management System) är detta heller inget som sätter upp hinder eller svårigheter.

> “For business data, there is a separate API layer to move data between different systems. Today there is integration with subscription systems like Kayak, Infosoft, Cprofit, Webabo and various integration platforms. More information about the APIs can be found here.” (*PayWay*, 2019)

Utöver det som slutkunderna *ser* finns det givetvis också system som tar hand om den “underliggande” prenumerationen. Dessa system har Payway både “inåtgående” och “utåtgående” integrationer mot för att underlätta där det behövs.

> “Tulo PayWay also includes an admin portal for customer service to manage customers and customer subscriptions. The work is simplified for customer service through self-service for the customer and the automation of tasks. In the admin portal, you can also easily package offers and manage campaigns.” - (*PayWay*, 2019)

En av de större “pelarna” av Payway är den s.k. administrationsportalen (Payway Administration Portal, eller PAP) där redaktionens kundtjänst får en bra inblick och goda möjligheter att sätta upp sina paket, kampanjer och kund prenumerationer på ett användarvänligt/intuitivt sätt.

Utöver detta, kan vi även från Payway egna webbsida få en idé över de tillgängliga/marknadsförda funktionerna. Vi hänvisar till figur 8.

Products & Pricing

Administrate and display your products and packages in an attractive way and make them easily accessible and easy to buy.

Integrations

No limits. PayWay is adaptive to integrations with existing systems, like subscription systems, customer databases, and open for external payment solutions.

Single Sign-On

The customer only needs to log in once to gain access to all systems. It's easy, comfortable and enables one-click shopping. (OAuth2)

Purchase Flow

The purchase flow, or "paywall", is the online shop. This is where customers register, activate and buy your products. An intuitive interface, safe payments through for example card and sms guarantees customer security.

My account

The customers own overview – showing the total engagement with you; account details, products, transactions and of course the possibility to administrate their information.

Statistics

Payway offers ready to use reports that can be exported to excel for follow up. For more advanced analytics we integrate with several Data Warehouse solution. Not least our own Tulo Engage Data Platform.

Figur 8: Del av de funktioner som Payway erbjuder och marknadsför i dagens läge. (PayWay, 2019)

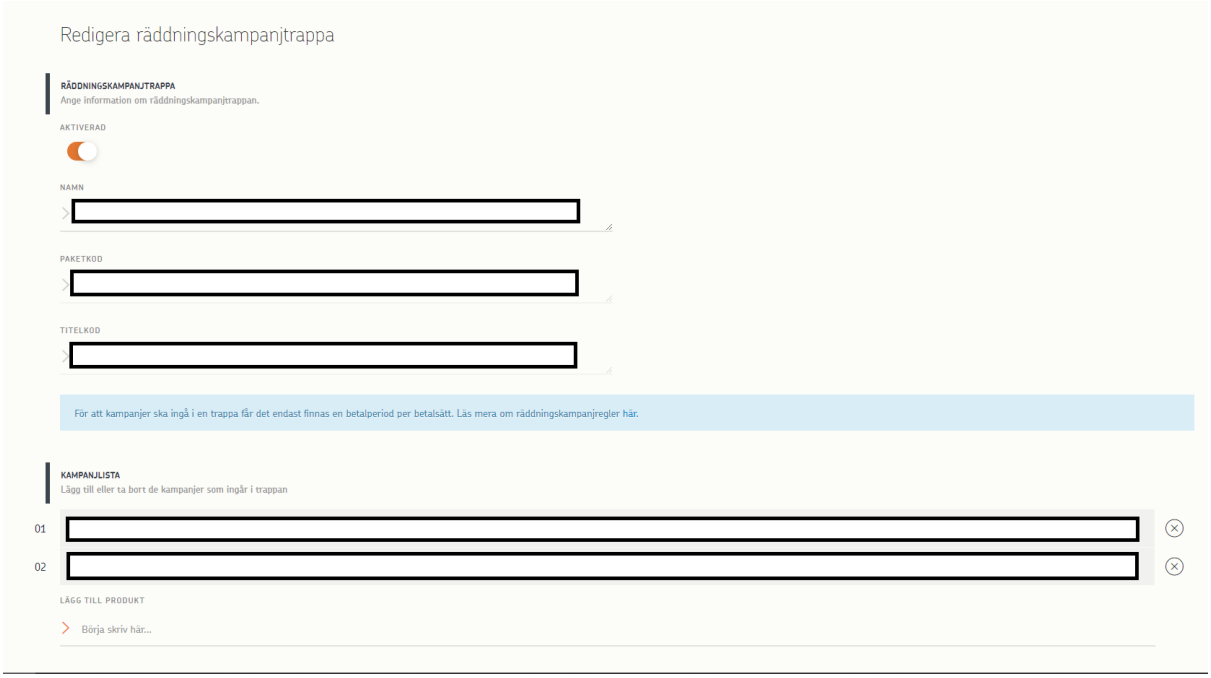
3.2 Kravspecifikation

Under räddningskampanjernas startfas ställdes ett antal krav på hur dessa skulle fungera (och vad som skulle vara möjligt). Vissa specifikationer blev under utvecklingens gång modifierade vartefter behov och vissa punkter kan även hålla mera "tyngd" än andra, men utvecklingsgrunderna för arbetet var något i stil med följande:

1. **Skapa räddningskampanjtrappor i PayWay**
2. **Redigera räddningskampanjtrappor i PayWay**
3. **Ta bort räddningskampanjtrappor i PayWay**

   Som tidigare nämnt, så är tankesättet bakom räddningskampanjer att dem ska fungera i "trappor". Det vill säga, när en kund går från en ordinarie produkt till första erbjudandet, ska det erbjudandet i regel vara "den bästa". Därefter kommer den näst-bästa, och så vidare.

   Utöver detta fanns även intresse i att kunna redigera och ta bort befintliga trappor, och inte vara påtvingade att skapa nya direkt en ändring ska genomföras.



Figur 9: Bild som visar/illustrerar hur uppsättningen av en räddningskampanjtrappa kan se ut i Payway.

4. Slå på / slå av räddningskampanjer i Payway

Utöver uppsatta räddningskampanjer och eventuella erbjudanden, behövdes ett sätt att smidigt kunna lägga igång eller stänga av allt vad heter räddningskampanjer. Huvudsakligen för att begränsa åtkomst och inte låta något nytt rusa fram för "snabbt", men även för att i framtiden kunna ha bättre kontroll ifall något skulle bli feltänkt.

5. Integration via API

Eftersom integrationer mot externa system är något som Payway i grund och botten håller som ganska viktig punkt, betyder det även att räddningskampanjer bör stödja detta. Detta är huvudsakligen relevant för när en slutkund bestämmer sig för att ge sig in på en räddningskampanj snarare än själva uppsättningen av trapporna.

6. Erbjuda räddningskampanjer till slutkund

Denna punkt kan anses som ganska självklar, men är oavsett en mycket viktig del. Efter att trapporna blivit påslagna och uppsatta, kommer vi vid varje uppsägning behöva göra en kontroll för att kontrollera ifall en räddningskampanj är uppsatt för det ordinarie paket som kunden håller på säga upp just nu. Blir svaret ja på denna fråga, behöver vi även implementera logik för att räkna ut vilket steg i trappan som kunden ligger på. Därefter kan ett erbjudande visas ut.

4. IMPLEMENTATION

Som vi redan itererat över är Räddningskampanjer en modul i ett befintligt system. Vi kan logiskt dela upp räddningskampanjer i tre olika delar:

4.1 Skapande av räddningskampanj

Administratörerna (redaktionen bakom tidningen) går in i Payway. I systemet sätts kampanjerna upp enligt de specifika behoven, som en produkt med egenskaper såsom namn, betal perioder och regler som redaktionen önskar för sin affärslogik. Därefter specificeras en konfiguration för "räddningstrappan" upp för att kunna erbjudas en slutkund.

Konfiguration av "räddningstrappan" innehåller information om vilka kampanjer som ska ingå i den "trappan", samt vilket paket (eller kampanj) som ska "utlösa" själva räddningskampanjen. Varje trappa sparas som ett dokument (i en dokument baserad databas) och innehåller information om vilket ordinarie paket som bör "utlösa" räddningskampanjen, vilka kampanjer som ska kunna erbjudas, samt en notering för att för administratören hålla reda på vilken "trappa" som är vilken i fall det finns ett större antal räddningskampanjtrappor. Dokumenten baserar sig på objektets unika ID, det vill säga numerisk identifiering som sammankopplar ett antal entiteter (olika tabeller i en databas), istället för att börja spara samma information på flera ställen.

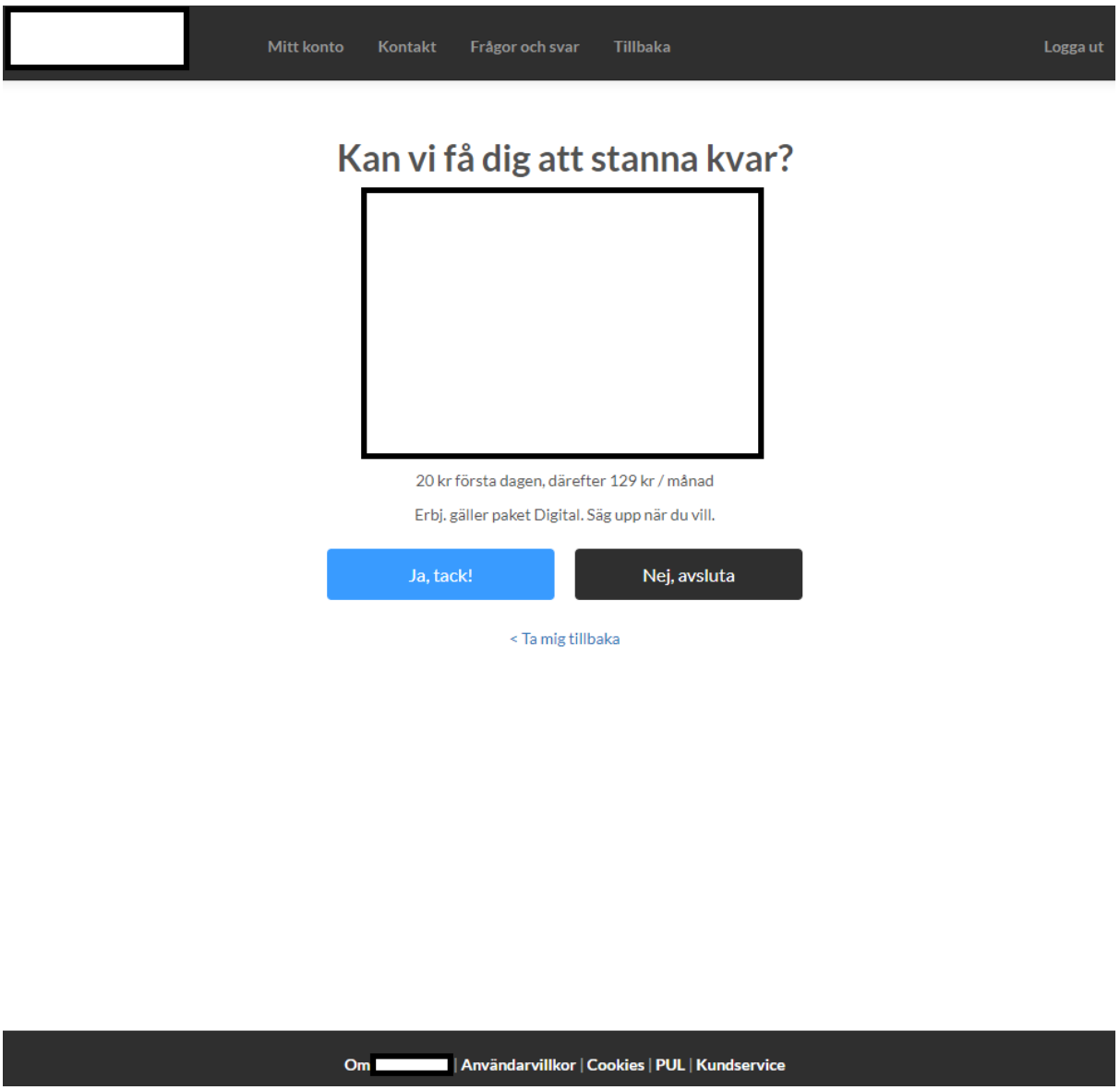
4.2 Uträkning av räddningskampanjerna för slutkund

När en slutkund går in i systemet och avslutar sin prenumeration, gör PayWay systemet först en kontroll mot de paket eller kampanj som användaren ligger på just nu. Ska kunden erbjudas en räddningskampanj? Är svaret ja, räknar vi ut vilket steg kunden ligger på i trappan och erbjuder den kampanjen som är konfiguerad. Finns det inget att erbjuda fortsätter avslutningsprocessen som vanligt, och kunden får (beroende på konfiguration) välja en orsak för avslutande och därefter slutföra uppsägningen.

4.3 Användning av räddningskampanjerna

Om slutkunden tackar ja till erbjudandet i “räddningstrappan”, får den det erbjudandet som tidigare presenterats, automatiskt uträknat, och det “steget i trappan” blir sedan det pris/produkt som säljs till kunden. Slurkunden får inget val i vilken kampanj, vilket steg i trappan som han/hon får, utan denna beräkning sker per automatik. Om kunden inte väljer det erbjudande som ges, avslutas prenumerationen.

Resultatet av detta är att redaktionen inte förlorat en kund, men snarare “vunnit tillbaka kunden”, så länge som den fortsatta/förnyade prenumeration som räddningskampanjen skapat är aktiv.

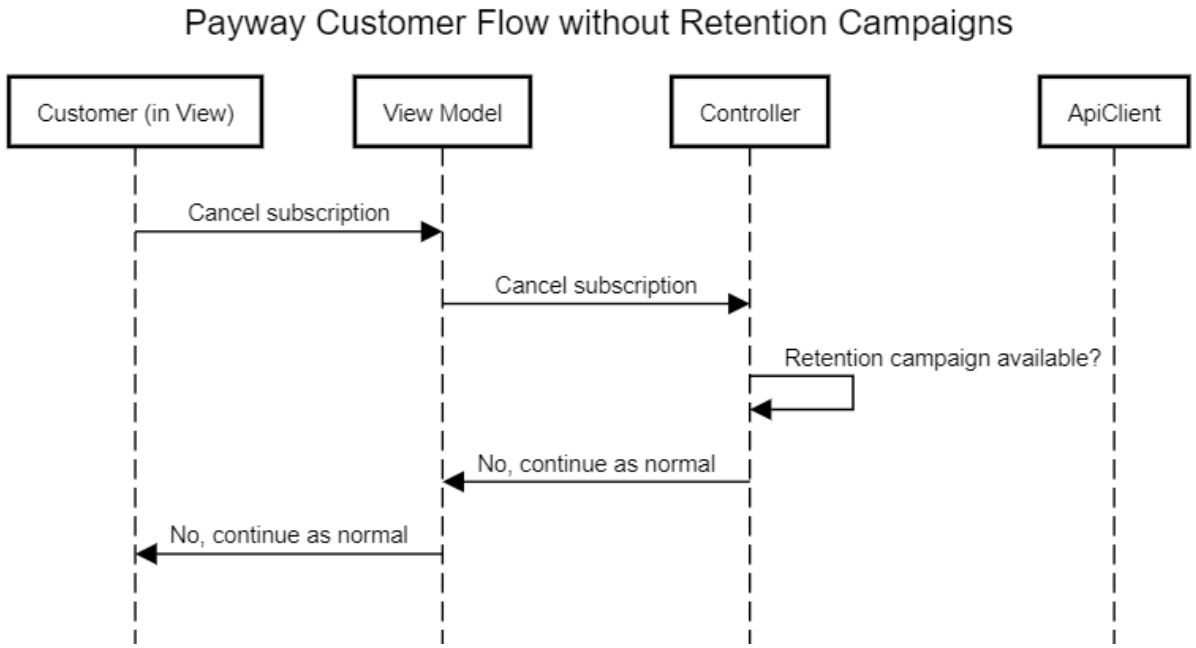


Figur 10: Illustration på hur ett räddningskampanjserbjudande kan se ut i Payway åt en slutkund. Information som kunde identifiera varken kund eller organisation har blivit ersatt med en blank ruta (se kravspecifikation 6).

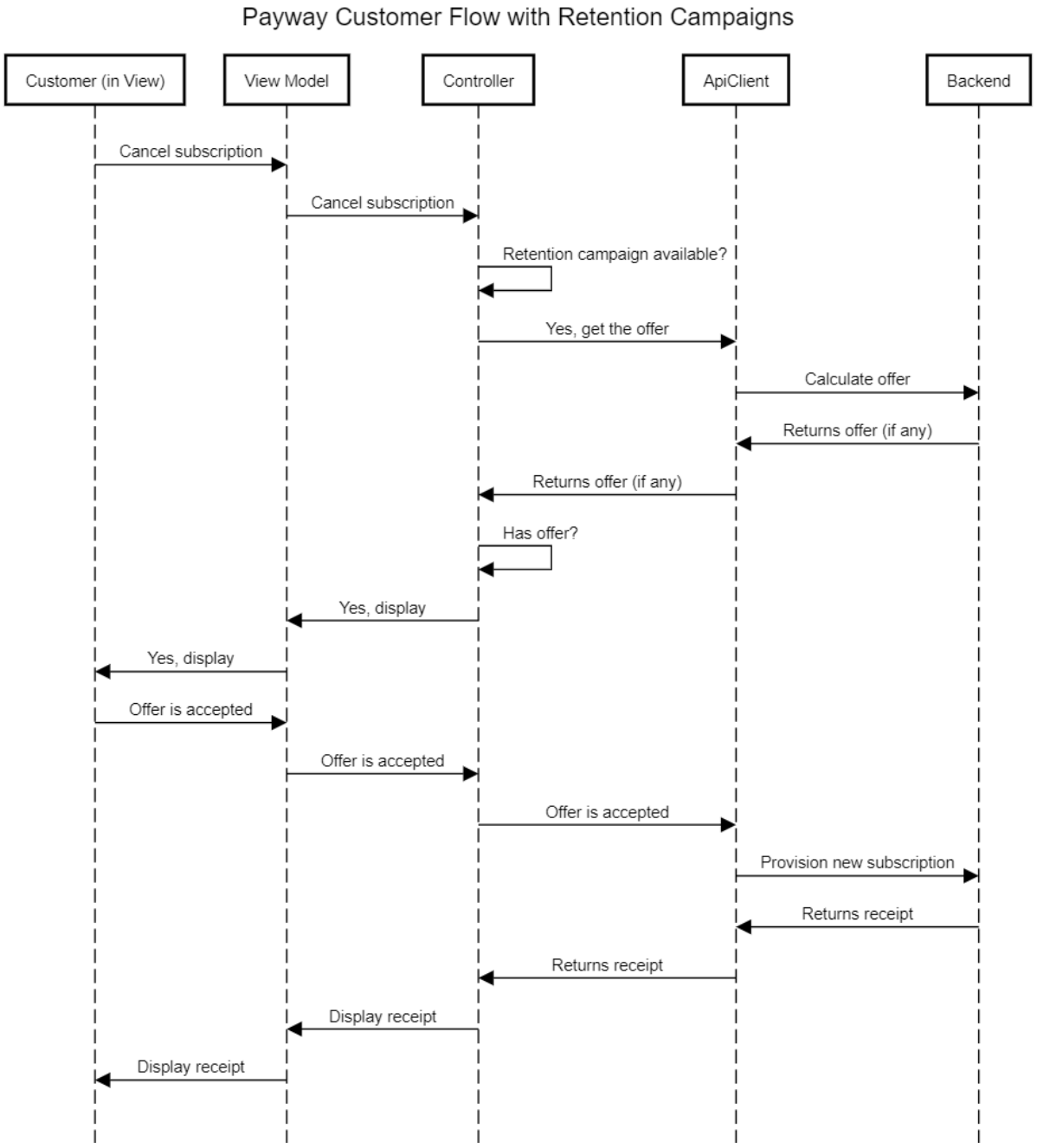
4.4 Server-generated content & flödet ur ett tekniskt perspektiv

Som tidigare nämnt används C# och MVC som arkitekturmodell för stort sett hela Payway. Detta innebär alltså att anrop utförs på "order" av kontrollerna, som sedan fyller ut den information som krävs i en s.k. vymodell, som därefter presenteras genom en vy till användaren i fråga. Detta gör med andra ord att renderingen tar plats serverside, och ifall dynamiskt innehåll är obligatoriskt på någon sida (dvs. innehåll som uppdateras *utan* att hela sidan behöver laddas om), behöver vyn presenteras på ett alternativt sätt. Sådana sätt kunde

vara genom att använda t.ex. AJAX (Asynchronous JavaScript And XML), som gör att man genom kod i JavaScript kan utföra anrop mot funktioner som är implementerade på Controller-nivå, och dra in/uppdatera information utan att behöva ladda om sidan. För att få en idé över hur detta flöde "går" refererar vi till figur 11:



Figur 11: Diagram över hur ett avslutningsflöde skulle kunna se ut, utan räddningskampanjer



Figur 12: Diagram över hur ett avslutningsflöde kan se ut med räddningskampanjer. Här bör det noteras ifall ett erbjudande inte hittas (“Has offer?”) eller ifall erbjudandet inte accepteras (“Offer is accepted”), går vi automatiskt tillbaka till det vanliga avslutningsflödet.

5. Resultatet av räddningskampanjer

Resultatet som man förväntar sig att räddningskampanjerna ska åstadkomma ligger, i grund och botten, på en finansiell nivå, eftersom det mer konkreta målet är att få existerande kunder att stanna kvar. Med det sagt ligger den estimerade räddningsgraden på cirka 15%. Av naturliga skäl kunde man förvänta sig att denna grad är exponentiell och progressivt blir högre vartefter räddningskampanjerna används.

Slutkundens vinst i detta är att han/hon får vara fortsatt prenumerant, men till ett betydligt bättre pris. Detta pris avgörs av användarna av systemet och varierar helt beroende på hur detta är uppsatt, men här löns det att komma ihåg det självklara: Ett bättre erbjudande betyder än bättre upplevelse för slutkunden, men även en mindre vinstmarginal för just den organisationen. Ett något sämre erbjudande kan fortfarande hålla kvar kunden (men givetvis med en mindre chans), men betyder även en större "vinst per kund".

För att lägga detta i perspektiv: En organisation som snittar 10.000 uppsägningar per år, skulle mitt i allt "endast" ha 8500 uppsägningar enligt det upplagda estimatet. Ifall det erbjudande som kunden erbjuds vid uppsägning kostar 9.90€ / månad, så betyder detta *(1500 * 9.90€ = 14.850€)* i en ytterligare vinst. Skulle erbjudandet handla om något billigare, säg ungefärligen halva priset, skulle den ytterligare vinsten landa på *(1500 * 4.90€ = 7350€)*.

6. FRAMTIDEN OCH REFLEKTION AV RÄDDNINGSKAMPANJER

Personligen gillar jag konceptet av räddningskampanjer, eftersom det inte är något “invasivt” men något som först visar sin existens när man går för att avsluta en viss prenumeration. Det är i dagens läge svårt att säga *precis* hur bra eller dålig utkomst alla berörda parter fått och kommer få av räddningskampanjerna, men jag anser att resultatet varit givande, givetvis från ett finansiellt perspektiv men även från ett kunskapsperspektiv. Som utvecklare har detta delsystem både fodrat frontend- och backend kunskap av mig samt förbättrat min nuvarande kunskap som dem är, och eftersom det dessutom var ett väldigt “ihophängande” projekt anser jag även att jag fick ett visst nöje av att vara med i utvecklingen av detta system.

Vad gäller den framtida utvecklingen av räddningskampanjer har jag svårt att tro att det är något delsystem som kommer komma mycket längre (med undantag för eventuella buggfixar och dylikt), men som med mycket annat är det svårt att säga vad framtiden sist och slutligen har att erbjuda.

KÄLLFÖRTECKNING

Advantages and Disadvantages of PHP. (2020, November 5). GeeksforGeeks. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-php/

Arora, S. K., Bordoloi, J., Rudra, Everton, C., Stefan, Jake, Bonron, & Arthur. (n.d.). *C# vs Java: Differences you should Know*. Hackr.io. Retrieved February 9, 2022, from https://hackr.io/blog/c-sharp-vs-java

Blackboard system. (n.d.). Retrieved February 5, 2022, from https://en.wikipedia.org/wiki/Blackboard_system

Danylko, J. (2017, July 13). *Top 10 Websites Written Using ASP.NET MVC*. Dzone.com; DZone. https://dzone.com/articles/top-10-websites-written-using-aspnet-mvc

Data-Centered Architecture. (n.d.). Retrieved February 5, 2022, from https://www.tutorialspoint.com/software_architecture_design/data_centered_architecture.htm

Deremuk, I. (2021, November 10). *C# vs Python - Pick the Right Programming Language For Your Project*. Litslink. https://litslink.com/blog/csharp-vs-python-choosing-right-language-for-your-project

Difference Between Bitbucket and GitHub. (2020, August 20). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-bitbucket-and-github/

Difference between HTML and HTML5. (2018, September 17). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-html-and-html5/

Do you know any examples of a PAC design pattern? (n.d.). Stack Overflow. Retrieved February 5, 2022, from https://stackoverflow.com/questions/73538/do-you-know-any-examples-of-a-pac-design-pattern

Ferrara, A. (2014, November 24). *Alternatives To MVC*. Ircmaxell's Blog. https://blog.ircmaxell.com/2014/11/alternatives-to-mvc.html

Gatlan, S. (n.d.). *RubyMine 2021.3.2 Build 213.6777.43 (Mac) - Download*. Softpedia. Retrieved

April 27, 2022, from https://mac.softpedia.com/get/Developer-Tools/RubyMine.shtml

House, T. (2020, September 10). *Laravel vs Ruby on Rails vs Symfony vs CodeIgniter vs CakePHP vs Yii vs Laminas (Zend) vs Django vs CubicWeb — Pros and Cons*. Dev Genius. https://blog.devgenius.io/laravel-vs-ruby-on-rails-vs-symfony-vs-codeigniter-vs-cakephp-vs-yii-vs-laminas-zend-vs-django-96726376b9a3

Landis, T. (n.d.). *Customer Retention Marketing vs. Customer Acquisition Marketing | OutboundEngine*. Retrieved February 6, 2022, from https://www.outboundengine.com/blog/customer-retention-marketing-vs-customer-acquisition-marketing/

Omelia, S. (2020, May 10). Ruby on Rails vs .NET Core: What Should You Choose? *Inverita*. https://inveritasoft.com/blog/ruby-on-rails-vs-net-core-what-should-you-choose

PayWay. (2019, June 5). Tulo. https://worldoftulo.com/payway/

Sudo Null Company. (n.d.). *Visual Studio 2019 Released*. SudoNull. Retrieved April 27, 2022, from https://sudonull.com/post/30532-Visual-Studio-2019-Released-JUG-Ru-Group-Blog