TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Vladimir Nitsenko 775488IDDR

CREATING A PROTOTYPE FOR THE PAYMENT METHOD AGGREGATOR

Diploma thesis

Supervisor: Toomas Lepikult
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Vladimir Nitsenko 775488IDDR

MAKSEMEETODITE AGREGAATORI VEEBIRAKENDUSE PROTOTÜÜBI LOOMINE

Diplomitöö

Juhendaja: Toomas Lepikult
PhD

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Vladimir Nitsenko

14.04.2021

Abstract

This diploma thesis is about analysing and creating a prototype of the new web application for a company providing payment method aggregator service. The problems that need to be solved in the new application comparing to an old one are bad performance, high infrastructure costs, slow and expensive scalability, outdated user interface and poor usability.

The main purpose of this thesis is to analyse possible solutions, bring functional and non-functional requirements and create a functional prototype of web application that can handle the forementioned complexities and provide reliable up-to-date user experience.

In the current diploma thesis, the architecture of the existing system and software deficiencies were described, the requirements of the new application analysed, the suitable technology was chosen, and the functional prototype created. The results of prototype evaluation are described and possible improvements to the system functionality and security are presented.

The outcome of the diploma thesis is considered successful as the offered prototype provides solutions for the highlighted problems, as well as possibility to use the prototype as a base and continue working on the web application to create a fully commercially profitable product, that implements both customer needs and appropriate security and flexibility requirements.

This thesis is written in English and is 37 pages long, including 8 chapters, 20 figures and 2 tables.

Annotatsioon

Maksemeetodite agregaatori veebirakenduse prototüübi loomine

Antud diplomitöö käsitleb uue veebirakenduse prototüübi analüüsimist ja loomist maksemeetodite koondamisteenust pakkuva ettevõtte jaoks. Probleemid, mis tuleb uues rakenduses lahendada võrreldes vanaga, on ebapiisav jõudlus, suured infrastruktuurikulud, aeglane ja kallis mastabeeritavus, aegunud kasutajaliides ja kehv kasutatavus.

Selle lõputöö põhieesmärk on analüüsida võimalikke lahendusi, tuua funktsionaalsed ja mittefunktsionaalsed süsteemi nõuded ja luua veebirakenduse funtsionaalne prototüüp, mis suudab toime tulla eelmainitud ülesannetega ja pakub usaldusväärset ajakohast kasutuskogemust.

Lõputöö käigus oli kirjeldatud olemas oleva süsteemi arhitektuur ja kaardistatud tarkvara kitsaskohad, analüüsitud uue veebirakenduse nõuded, valitud sobiv tehniline lahendus ja loodud veebirakenduse prototüüp. Samuti on töös käsitletud prototüübi testimise tulemusi ning toodud võimalikud süsteemi funktsionaalsuse ja turvalisuse parendused tulevikus.

Disainitud süsteemi arhitektuur ja koostatud funktsionaalne prototüüp on aluseks arendustöö jätkamiseks täislahenduse loomise nimel ja kogu vajaliku äri- ja süsteemiloogika lisamiseks ja rakendamiseks.

Diplomitöö tulemus loetakse edukaks, kuna loodud prototüüp pakub lahendusi esiletõstetud probleemidele ning võimaluse jätkata tööd veebirakenduse arendamisel, et luua täielikult ärikasutatav kasumlik toode, milles oleks arvestatud nii klientide soovidega kui tänapäeva turvalisuse ja paindlikkuse nõuetega.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 37 leheküljel, 8 peatükki, 20 joonist, 2 tabelit.

List of abbreviations and terms

API	Application Programming Interface, an interface that defines interactions between multiple software applications
.NET	Cross-platform open source developer platform for building different types of applications
3D Secure	A protocol designed to be an additional security layer for online credit and debit card transactions
AJAX	Asynchronous JavaScript and XML, used for data exchange with the server and updating the page without having to refresh it
B2B	Business-to-Business
Back-end	Application and database that work on the server to deliver information to the user
Bearer token	A cryptic string, usually generated by the server in response to a login request
CI/CD	Combined practices of continuous integration and continuous delivery of the software
COVID-19	Coronavirus disease SARS-CoV-2
CSS	Cascading Style Sheets, the language used to style an HTML document
CVC	Card Verification Code, security code located on the payment card
Development roadmap	Initiatives, epics and features in the software engineering pipeline, a high-level snapshot of project major objectives
DevOps	A set of practices that combines software development and IT operations
Endpoint	The location from which it is possible to access the remote server resources
End-user	A person who ultimately uses or is intended to ultimately use a product
Front-end	Application that works in user's Web browser and refers to UI
GET	HTTP request type to retrieve a resource from the server
HTML	HyperText Markup Language, the standard markup language for Web pages

HTTP	Hypertext Transfer Protocol, a protocol which allows the fetching of network resources
HTTPS	Hypertext Transfer Protocol Secure, signals the browser to use an added encryption layer to protect the network traffic
Idempotent / Idempotency	HTTP method is idempotent if an identical request can be made several times in a row while leaving the server in the same state; an idempotent method does not have any side-effects (except for keeping statistics)
IaaS	Infrastructure as a service
IBAN	International Bank Account Number
iframe	HTML iframe is used to display a web page within a web page
IT	Information Technology
JavaScript / JS	Web programming language
JSON	JavaScript Object Notation, a lightweight data-interchange format
JSONP	JSON with Padding, enables sharing of data bypassing same-origin policy, which disallows running JavaScript code fetched from outside the page's originating site
MVC	Model-View-Controller, a software design pattern commonly used for developing user interfaces
npm	The package manager for the Node JavaScript platform, provides modules and manages dependencies
OAuth	Industry-standard protocol for authorization
On-premise software / server	Installed and runs on computer(s) on the premises of the organization using the software/server
ORM	Object-Relational Mapping, programming technique for converting data between relational databases and object-oriented programming languages
PaaS	Platform as a service
PCI DSS	Payment Card Industry Data Security Standard
RAM	Random-access memory, a form of computer memory that can be read and changed in any order, typically used to store working data and machine code
REST	Representational State Transfer, architectural style for providing standards for information exchange
RESTful application	An application that follows REST style and principles
RSA	RSA (Rivest–Shamir–Adleman) is an algorithm used to encrypt and decrypt messages
SaaS	Software as a service

Service-level agreement / SLA	A commitment between a service provider and a client
SOAP	Simple Object Access Protocol, a messaging protocol specification for exchanging structured information
SQL	Structured Query Language, a domain-specific language designed for managing data in a relational database
TLS/SSL	The standard technology that uses encryption for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems
UI	User interface
URL	Uniform Resource Locator, a web address
WCAG	Web Content Accessibility Guidelines, a set of recommendations for making Web content more accessible for people with disabilities and user-agents
Web Messaging	API allowing documents in Web browser to communicate with one another across different source domains, e.g. between a parent document and an iframe
Widget	A web page or web application that is embedded as an element of a host web page, but which is substantially independent of the host page, having limited or no interaction with the host

Table of contents

List of figures

List of tables

1 Introduction

The growth of internet users has accelerated dramatically in last two decades. This is driven by Internet connection cost affordability and rapid development of web applications and services that make our life easier and help to save one of our most valuable resources – time. According to statistics of World Bank percentage of the population using internet grew from 6.5% in 2000 to almost 50% in 2017 [1].

Since the start of COVID-19 pandemic, even if that has occurred within a context of stagnant or contracting overall retail, the market share of online retail against offline has increased more rapidly all over the world [2]. This is a strong sign of ever-increasing digitalization, that makes customers more and more comfortable using Internet services and e-commerce solutions.

E-commerce providers compete between each other to make their web applications faster and simpler, improve intuitiveness and usability, and provide stability and security. User-friendly and reliable web service is more attractive for users and potential customers, so it is in direct correlation with image, profits and overall success of an enterprise.

1.1 The background

The company X that provides B2B solutions has a variety of products for web marketplaces. There are out-of-the-box configurable applications that help to organize the checkout process for retail customers of e-stores. The web application, that is viewed in this thesis and needs to be reworked, offers the functionality of payment method aggregator. This is a service through which e-commerce merchants can process their payment transactions. The solution allows merchants to accept debit and credit cards, bank transfers, mobile payments, e-wallets and offer slicing of the payable amount into parts for retail customers. Merchants can benefit from a set of payment methods but have only one direct integration with the payment aggregator, without having to setup a merchant account with a bank or card association and eliminating the need to maintain dozens of integrations with forementioned institutions. As payment aggregator processes and stores payment card data there is no need for a merchant to be PCI DSS compliant, because the aggregator is fully responsible this.

1.2 The problem and the goal

The existing payment method aggregator does not meet current web application standards anymore. There are significant downsides in performance and usability, which are described in detail in the next chapter. It is worth mentioning that similar payment solutions are offered by wide range of competitors. In order to maintain the attractiveness of the payment method aggregator solution offered by the company X it was decided to rewrite the web application front-end and back-end parts.

The goal of this thesis is to analyse and choose software development technology stack and infrastructure, and to create a prototype for the new web application. The main purpose of the prototype is to provide a solution so that payment methods widget will load quickly, and UI will be intuitive for the end-user, application will be capable of handling high loads and provide security according to market standards and industrial requirements. The prototype could then be taken as a baseline for further development and implementation of the full range of business and system logic.

1.3 Methodology

For prototyping and developing a new web application the approach must be systematic. This will help to design and build a software that not only meets the expectations of stakeholders and customers in efficient, cost-effective and quality way, but is also understandable for current and future developers and product owners.

In this diploma thesis the author analysed existing solution and its deficiencies, described functional and non-functional product requirements for the new application. A set of software technologies and infrastructure possibilities were analysed, and the most suitable programming language, frameworks and infrastructure platform was chosen based on their advantages, community size and current company engineers expertise.

Software prototyping is the creation of an incomplete version of the required software and typically it simulates a few aspects of the final solution. Author described architecture, mock-ups and data exchange examples of the new web application. The functional prototype was created for both front-end and back-end applications, and it provides the basic functionality and user interface, which will be further developed into

complete solution by the company team. Prototype was tested to satisfy product requirements in performance, WCAG compliance and UI usability. The summary of the results of prototyping a new payment methods aggregator web application finalizes the diploma thesis scope.

2 Existing solution description

The company and existing payment methods aggregator product considered in this thesis offers an individual approach to its customers, meaning that the product enhancements and improvements are developed according to merchants' needs. The product manager is in tight contact with company's customers – e-commerce merchants, figuring out and negotiating requirements for new features, and once agreed they are added to the development roadmap.

There are several possible alternative payment aggregator solutions provided by third parties. Some of them are PayPal, Worldpay, Ingenico, Amazon Payments, Braintree, Stripe, Adyen, CyberSource, Square, Authorize.net. The company customers are large and middle enterprises, which are not satisfied with existing market solutions and need exclusive request proposal approach. This is the reason why customers would like to continue using the product, rather than switch to one of the forementioned existing solutions.

There is a number of deficiencies in the current web application that need to be resolved with a new solution. There are functional and usability issues along with non-functional severities.

2.1 Current high-level architecture

The diagram in Figure 1 illustrates high-level architecture and main interaction points of the current system.

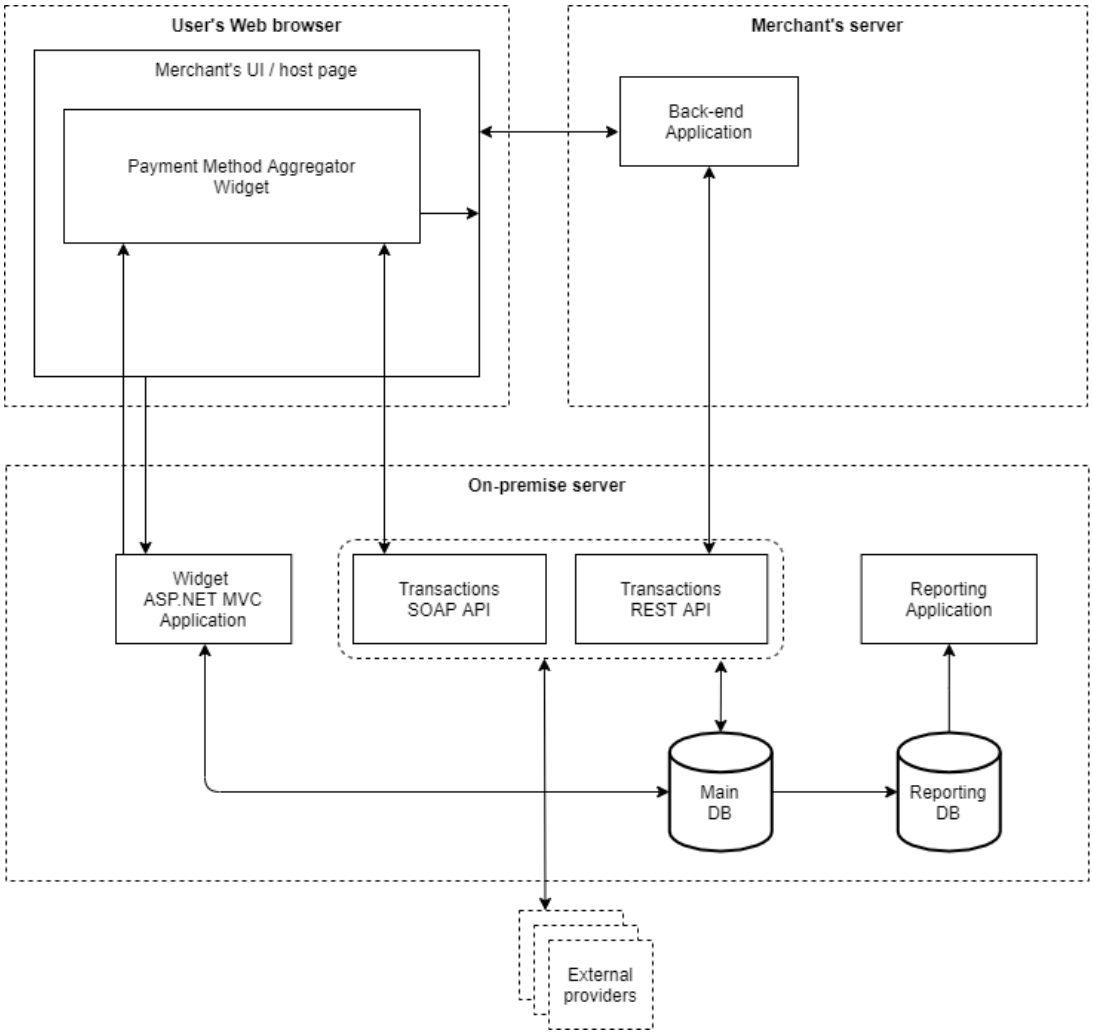


Figure 1. Existing high-level architecture.

2.2 Functional and usability deficiencies

- Widget layout is outdated: elements, fields and logos position and design are not following current market standards.
- There is no possibility to load merchant CSS to fully override default styles.
- There are too much manual actions for end-user: must select card brand, enter card holder name, switch between month and year fields for card expiry.
- Selection of the payment method is inconvenient, as user must click the small radio button, that is especially annoying while using small screen devices.

- Field values entered by the user are not validated on the fly, but only when user clicks Continue or Pay button.
- UI checksum validation of payment card or IBAN numbers is missing.

2.3 Non-functional deficiencies

- Application uses JSONP technique, which is vulnerable to the data source replacing the innocuous function call with malicious code [3].
- Back-end service is built to use SOAP technology for data exchange. Comparing with REST architecture, SOAP is not easily scalable, it has worse performance and data is not cacheable, also output data is restricted to XML only.
- Merchant configuration data is loaded from the database and processed, and external services requests are performed while loading the widget, this means end-user waiting can be long enough due to server-side execution time. The time that user is waiting for a payment page to be shown has critical importance, it should be as fast as possible, otherwise transaction can be abandoned by the buyer.
- Application is hosted at physical infrastructure, that means slow and expensive scalability, as servers' capacity cannot be always increased on demand and we need to wait for example for delivery of an additional RAM.
- Current application and its libraries are created in older versions of .NET framework and use older third-party libraries, some of them cannot be updated without a major code refactoring.
- In the next few years, a system can potentially become a legacy one, this would mean higher maintenance costs and lower software engineers motivation to work on that project.
- The look of current front-end is outdated, which can raise doubts in the online store customer, transaction can be abandoned.

3 Product requirements

The software requirements specification consists of functional and non-functional requirements. It describes features and functionalities of the target system. It may also include a set of use cases which describes user interactions that the software must provide to the user for perfect usability.

Functional requirements provide a list of rules about how a product must behave, description of its features and functions. Non-functional requirements can also be called "quality attributes", they describe operation capabilities and constraints of the system. Service-level agreement is often a part of non-functional requirements.

3.1 Functional requirements

- Widget loading time must be below 200ms.
- User must have an option to scan his payment card to simplify entering of card number and expiry.
- Card type must be determined automatically from the card number.
- Card holder's name must be prefilled from shopping cart billing address.
- Web page should be WCAG 2.0 compliant – readable by screen readers so that disabled people can use the widget.
- Input fields must be validated and not allow illegal characters.
- Payment card and IBAN numbers must be validated in UI using their checksum.
- Wrong user inputs must be highlighted with a text below the invalid field.
- UI must support translations depending on user's browser language settings.
- UI must display information about the status and loading indicator while user is waiting for action to be processed.
- Widget styles must be customizable via custom merchant CSS.
- There must be a checkbox to save payment card details, so user does not have to fill them next time.

3.2 Non-functional requirements

- UI must have responsive design and fit well on various desktop and mobile devices.

- Application must be stable not less than for 1000 concurrent users.
- Data exchange between client and server must be done using HTTPS.
- OAuth2.0 authorization must be implemented in back-end service.
- Queries from client to server must be done via asynchronous calls without reloading the widget.
- Application must be accessible via REST.
- Sensitive data (card number, CVC) must be encrypted for transferring via network.
- Customer payment card data must be securely stored in the database.
- All data validation must be performed in both UI and server-side.

4 Software development technologies

There are more than 270 popular programming languages [4] that have been used significantly in the near past or are widely spread nowadays and accurate in software development. It can be a challenge when it comes to choosing technologies to be used in the new web application. Every technology and programming language has its own specification, community support, advantages and disadvantages.

According to Stack Overflow[1] developer survey, that is conducted yearly from 2011 [5], the 10 most used technologies among professional developers in the year 2020 were JavaScript, HTML/CSS, SQL, Python, Java, Bash/Shell/PowerShell, C#, TypeScript, PHP and C++ [6]. The most popular back-end programming languages from them are Java, Python, C++ and C# [7]. C++ will not be considered in this scope as it is not commonly used for web application development, but rather for games, device drivers, high-performance scientific software, embedded programs, windows client applications, libraries and compilers for other programming languages [8].

A framework is software that provides an extensible and standardized way to build and deploy applications. It is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate the development of software applications, products and solutions. Depending on the framework, they may include tools, compilers, libraries, support programs, APIs and other components to enable easy and standardized development of an application or system.

4.1 Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers “write once, run anywhere”

[1] Stack Overflow is a public platform located at www.stackoverflow.com that serves 100 million developers and technologists every month, making it one of the 50 most popular websites in the world [9].

meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. The syntax of Java is similar to C and C++ but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities such as reflection and runtime code modification [10].

4.1.1 Spring

The most popular Java web frameworks are Spring, GWT and JSF [11], from which Spring Framework is the most widely used by professional developers [12]. There are Spring Boot, Spring MVC and Spring Cloud components of Spring framework available.

Due to scalability and microservices requirement for the new web application the Spring Cloud can be considered as possible primary technology to use. The main advantages and disadvantages of this are described as follows.

Advantages and features [13]:

- distributed/versioned configuration;
- service registration and discovery;
- routing;
- service-to-service calls;
- load balancing;
- Circuit Breakers;
- global locks;
- leadership election and cluster state;
- distributed messaging;
- integrates well with popular PaaS providers like Cloud Foundry, Amazon Web Services and Azure [10].

Disadvantages [14]:

- can be only used with Java – software developers in the team would first need to learn Java and then Spring Cloud, which creates a high learning curve;
- sometimes it adds unnecessary dependencies during deployment which increases its binary size;

- lack of content in the official documentation.

4.2 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed. Since there is no compilation step, the edit-test-debug cycle is incredibly fast [15].

The most popular web development frameworks for Python as of the year 2020 are Django and Flask [16].

4.2.1 Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. It takes care of much of the hassle of web development, so a developer can focus on writing the application without needing to reinvent the wheel – with the help of available packages. Django framework is free and open source, has a thriving and active community, great documentation and many options for free and paid-for support. Django is a thriving, collaborative open source project, with many thousands of users and contributors. It has evolved into a versatile framework that is capable of developing any type of website [17]. The main advantages and disadvantages of this are described below [18].

Advantages:

- the main focus is on the fast development of complex and large projects;
- scalability, security;
- ORM support for MySQL, Oracle, Postgre and NoSQL databases;
- it has own internationalisation system for supporting multilingual websites;

- built-in support for Caching, AJAX, forms;
- there is administration interface to maintain user activities;
- testing is made easy with the use of a lightweight web server.

Disadvantages:

- software developers in the team would first need to learn Python and then Django, which creates a high learning curve;
- it is too monolithic framework;
- everything in it is based upon ORM.

4.2.2 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself [19]. The main advantages and disadvantages of this are listed as follows [20].

Advantages:

- scalability;
- simple development;
- flexibility;
- modularity, efficiency, testability;
- performance.

Disadvantages:

- software developers in the team would first need to learn Python and then Flask, which creates a high learning curve;
- not standardized;
- few built-in tools, need to have many custom components and extensions;

- all developers should have very high competence level to be able to produce hight quality code as Flask is not standardized.

4.3 C#

C# (pronounced "see sharp") is a general-purpose, multi-paradigm programming language encompassing static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented class-based and component-oriented programming disciplines. C# was developed around 2000 by Microsoft as part of its .NET initiative [21]. C# has its roots in the C family of languages, so it is immediately familiar to C, C++, Java, and JavaScript programmers. Since its origin, C# has added features to support new workloads and emerging software design practices [22].

The most popular web application development frameworks that provide C# are ASP.NET MVC and ASP.NET Core [23].

4.3.1 ASP.NET MVC

ASP.NET MVC is a web application framework developed by Microsoft that implements the MVC pattern. It is no longer in active development with the last final release dated 28.11.2018 [24].

ASP.NET offers three frameworks for creating web applications: Web Forms, ASP.NET MVC and ASP.NET Web Pages. All three frameworks are stable and mature, and web applications can be created with any of them. These ASP.NET frameworks are based on the .NET Framework and share core functionality of .NET and ASP.NET. They offer a login security model based around membership, and all three share the same facilities for managing requests, handling sessions, and other parts of the core ASP.NET functionality [25].

4.3.2 ASP.NET Core

ASP.NET Core is the successor of ASP.NET MVC. It is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps. ASP.NET Core integrates seamlessly with popular client-side frameworks and libraries, including Blazor, Angular, React, and Bootstrap [26].

Stack Overflow queries rate, that is reflected in Figure 2, shows the rapid growth of ASP.NET Core popularity from the year 2016 when the first version of .NET Core was released. Trends also show the rapid declination in ASP.NET MVC questions amount on Stack Overflow platform. From the end of 2018 year ASP.NET Core questions share exceeded ASP.NET MVC with the last continuing to decline. Microsoft Corporation, the developer of the frameworks of .NET family, continues to actively work on and enhance .NET Core. There are already 7 versions of .NET Core released from 2016 [27]. The next major release of .NET Core following 3.1 is .NET 5. ASP.NET Core 5 is based on .NET 5, but retains the name "Core" to avoid confusing it with ASP.NET MVC 5 [28].

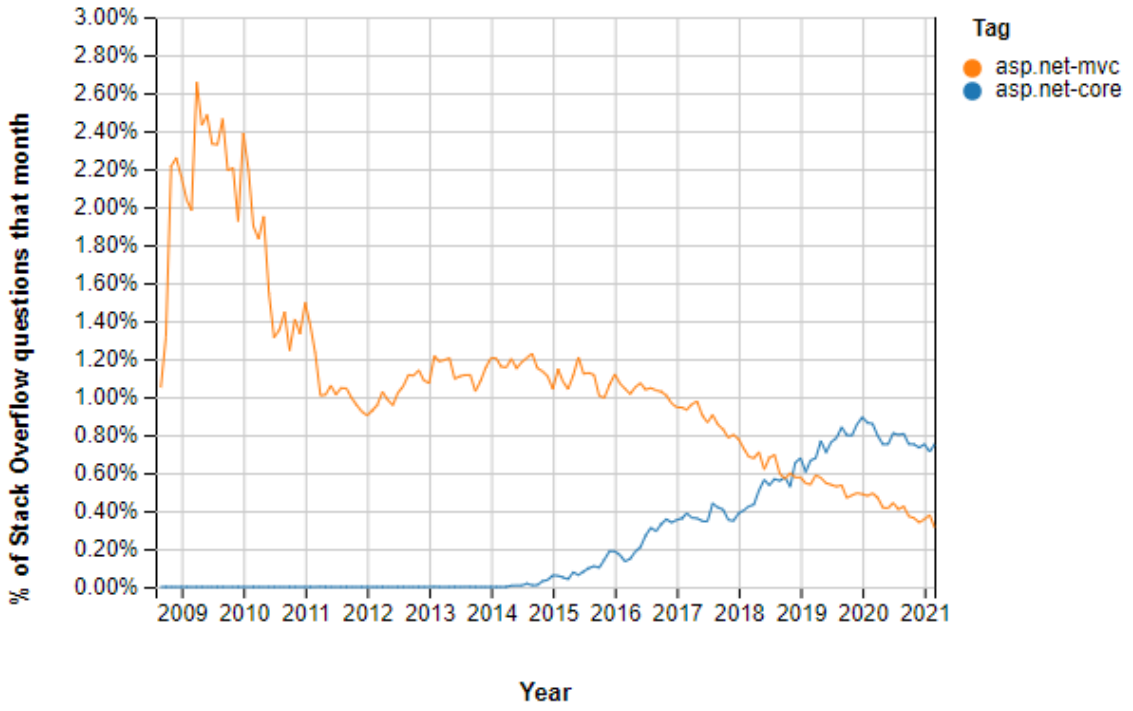


Figure 2. ASP.NET MVC and ASP.NET Core trends.

Advantages [26]:

- new Kestrel web server, higher performance than ASP.NET MVC;
- easy to develop cross-platform web applications – has built-in support;
- IDE Visual Studio available for Windows and Mac, lightweight Visual Studio Code can also be used;
- rapidly growing popularity and community;
- cloud-ready, supports microservices;

- modularity via NuGet, it manages to include only needed libraries;
- the code is open-source – increases community contribution and collaboration.

Disadvantages:

- software developers in the team are proficient with .NET Framework and ASP.NET MVC – the use of ASP.NET Core will require little to moderate time for additional learning;
- documentation of some new features or changes could be insufficient in comparison to mature and long-history ASP.NET MVC.

4.4 Front-end frameworks

Front-end is sometimes named also as client-side. This means that the code runs in a Web browser on the user's computer in order to provide enriched user interface interaction and dynamic data retrieval from the server. The front-end framework is usually a set of JavaScript libraries, that help creating applications in a standardized, modular and *strongly typed*[1] way. In order to decrease the time and bandwidth consumed by downloading the code and style files into user's browser, minification of the front-end application is very commonly used. This is the process of removing unnecessary characters, comments, and shortening function and variable names in JavaScript, CSS or HTML code without changing the way that it works.

From the perspective of front-end application development, the most popular and widely used frameworks among professional software developers are jQuery, React.js, Angular, ASP.NET, Express, ASP.NET Core, Vue.js, Spring, Angular.js, Flask and Django. Also Node.js, Pandas and React Native are very commonly used together or in addition to the mentioned frameworks [6].

The designed application will have minimal front-end capabilities, there is a need for secure and stable, but lightweight and simple approach.

---

[1] Strongly typed programming language is the one where variables and other data structures can be declared to be of a specific type, like a string or a boolean, and there are strict checks of the validity of their values. This isn't possible in pure JavaScript, which is loosely typed.

4.5 Infrastructure

Information technology infrastructure is a combination of hardware and software components needed for the operation and management of enterprise IT services and environments. There are various ways of organizing infrastructure that will provide required capabilities and support deployment, running and monitoring the application.

First of all, there is an option to create and maintain infrastructure on-premise. This means servers are owned by the development company and it will be responsible for establishing, configuring and maintaining them.

Another option is to delegate infrastructure management to the professional third-party – an enterprise which, for overall effectiveness, specializes on providing and maintaining essential equipment, components, data, policies, processes, human resources and external contacts.

The third option that is considered in this thesis is Cloud Computing. Various types of clous services, like IaaS, PaaS, SaaS or even private clouds, are continuously growing and according to the forecast for 2022 Cloud Computing industry size will double in comparison to the year 2018 [29]. Cloud infrastructure and services are commonly provided out-of-the-box, so it takes much less resources and effort to prepare infrastructure and connect services according to needs of specific project or enterprise.

Web applications have to be quickly scalable depending on the number of concurrent users and the load. It should be possible to deploy infrastructure and applications, adjust capacity and performance, connect monitoring tools and make changes rapidly and cost-saving. Because of fluctuating and increasing web services and e-commerce demand due to COVID-19 [30], forementioned is especially important to maintain competitiveness in current market realities.

4.6 The choice of technologies

The software development company has implemented project policies, that include development practices, programming languages and infrastructure providers. The current solution is developed using ASP.NET MVC on C#. The team consists of senior

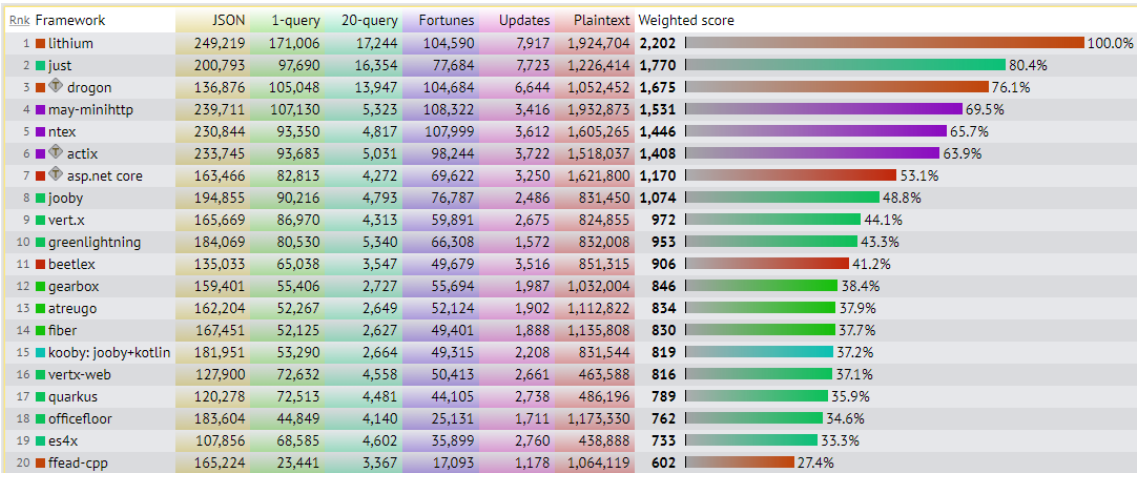
engineers that specialize in C# and .NET development with the focus on back-end engineering.

Even though C# stands on the third place among the three most popular back-end programming languages, it is chosen to be the web service programming language for the new web application. The reason for that are company policies, existing team engineering expertise, and limited human and time resources, that would make new technologies high learning curve unreasonable. Summarizing all these facts it is rational to continue using C# and .NET also for the new application.

For serving the front-end part of web application Node.js with a set of npm packages will be used as it provides modularity and is supported by a large developer community. For automation of common tasks, like minification or testing, Gulp will be used. The company has engineers that specialize in both frameworks so it will be straight forward to smoothly continue working on the project after prototype is ready.

Cloud infrastructure will be used to host both front-end and back-end of the designed web application, as there is a need for quick scalability due to varying seasonal loads, and pricing model that will depend on consumed capacities. Rapid infrastructure deployment and ongoing configuration can be supported by DevOps department that already exists in the company. According to company policies Microsoft Azure Cloud will be used as IaaS, SaaS and PaaS provider.

The chart in Figure 3 shows the results of cloud performance tests for various frameworks [31]. ASP.NET Core is located on the 7-th place in composite performance scores table. Despite there are more performant frameworks, ASP.NET Core is the only one from forementioned mostly used, that we can see between the 20 best web frameworks in the year 2021. This confirms the sustainability and top performance that is extremely essential in the century of ever evolving technologies. The new ASP.NET Core application will incorporate REST and requests and responses will be handled in JSON format.



Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score
1	lithium	249,219	171,006	17,244	104,590	7,917	1,924,704	**2,202** 100.0%
2	just	200,793	97,690	16,354	77,684	7,723	1,226,414	**1,770** 80.4%
3	drogon	136,876	105,048	13,947	104,684	6,644	1,052,452	**1,675** 76.1%
4	may-minihttp	239,711	107,130	5,323	108,322	3,416	1,932,873	**1,531** 69.5%
5	ntex	230,844	93,350	4,817	107,999	3,612	1,605,265	**1,446** 65.7%
6	actix	233,745	93,683	5,031	98,244	3,722	1,518,037	**1,408** 63.9%
7	asp.net core	163,466	82,813	4,272	69,622	3,250	1,621,800	**1,170** 53.1%
8	jooby	194,855	90,216	4,793	76,787	2,486	831,450	**1,074** 48.8%
9	vert.x	165,669	86,970	4,313	59,891	2,675	824,855	**972** 44.1%
10	greenlightning	184,069	80,530	5,340	66,308	1,572	832,008	**953** 43.3%
11	beetlex	135,033	65,038	3,547	49,679	3,516	851,315	**906** 41.2%
12	gearbox	159,401	55,406	2,727	55,694	1,987	1,032,004	**846** 38.4%
13	atreugo	162,204	52,267	2,649	52,124	1,902	1,112,822	**834** 37.9%
14	fiber	167,451	52,125	2,627	49,401	1,888	1,135,808	**830** 37.7%
15	kooby: jooby+kotlin	181,951	53,290	2,664	49,315	2,208	831,544	**819** 37.2%
16	vertx-web	127,900	72,632	4,558	50,413	2,661	463,588	**816** 37.1%
17	quarkus	120,278	72,513	4,481	44,105	2,738	486,196	**789** 35.9%
18	officefloor	183,604	44,849	4,140	25,131	1,711	1,173,330	**762** 34.6%
19	es4x	107,856	68,585	4,602	35,899	2,760	438,888	**733** 33.3%
20	ffead-cpp	165,224	23,441	3,367	17,093	1,178	1,064,119	**602** 27.4%

Figure 3. Web application frameworks peak performance comparison.

There is a need for a new widget database, which would be the most performant to have it also in Azure. For this Azure SQL Database will be used. The code-first approach, which allows to define data model using C# classes and attributes, will simplify the development process, make further CI/CD easier and help to automate database schema changes via automatic migrations.

5 Prototype creation

Prototyping is the process of developing an incomplete working version of a product or system that has to be engineered. The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built [32].

The functional prototype is created in co-operation with the company product manager and business representatives of key company customers. The prototype model, displayed in Figure 4 and rounded with green, is used for prototype development. After the prototype version is accepted by product manager and the key company customers the development of the full product can be incorporated into the development roadmap and engineering team can continue working on the full implementation using iterative development model.

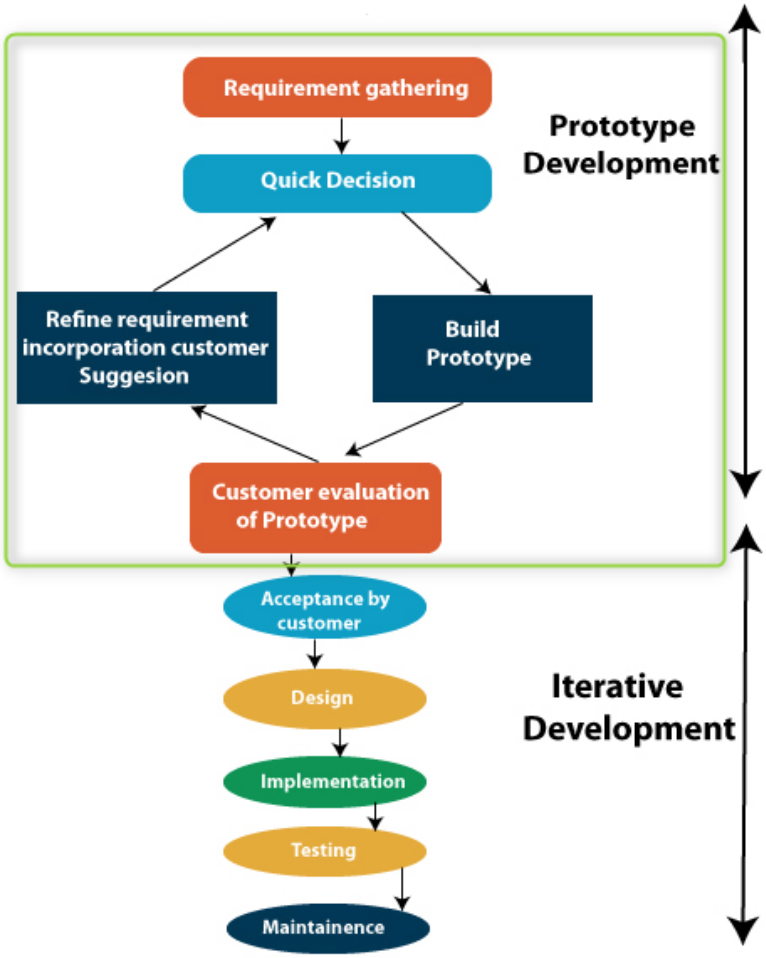


Figure 4. Prototype model steps.

5.1 High-level architecture

The Figure 5 diagram was created based on the analysis in previous chapters. It describes the high-level architecture of the system after the new web application will be implemented.

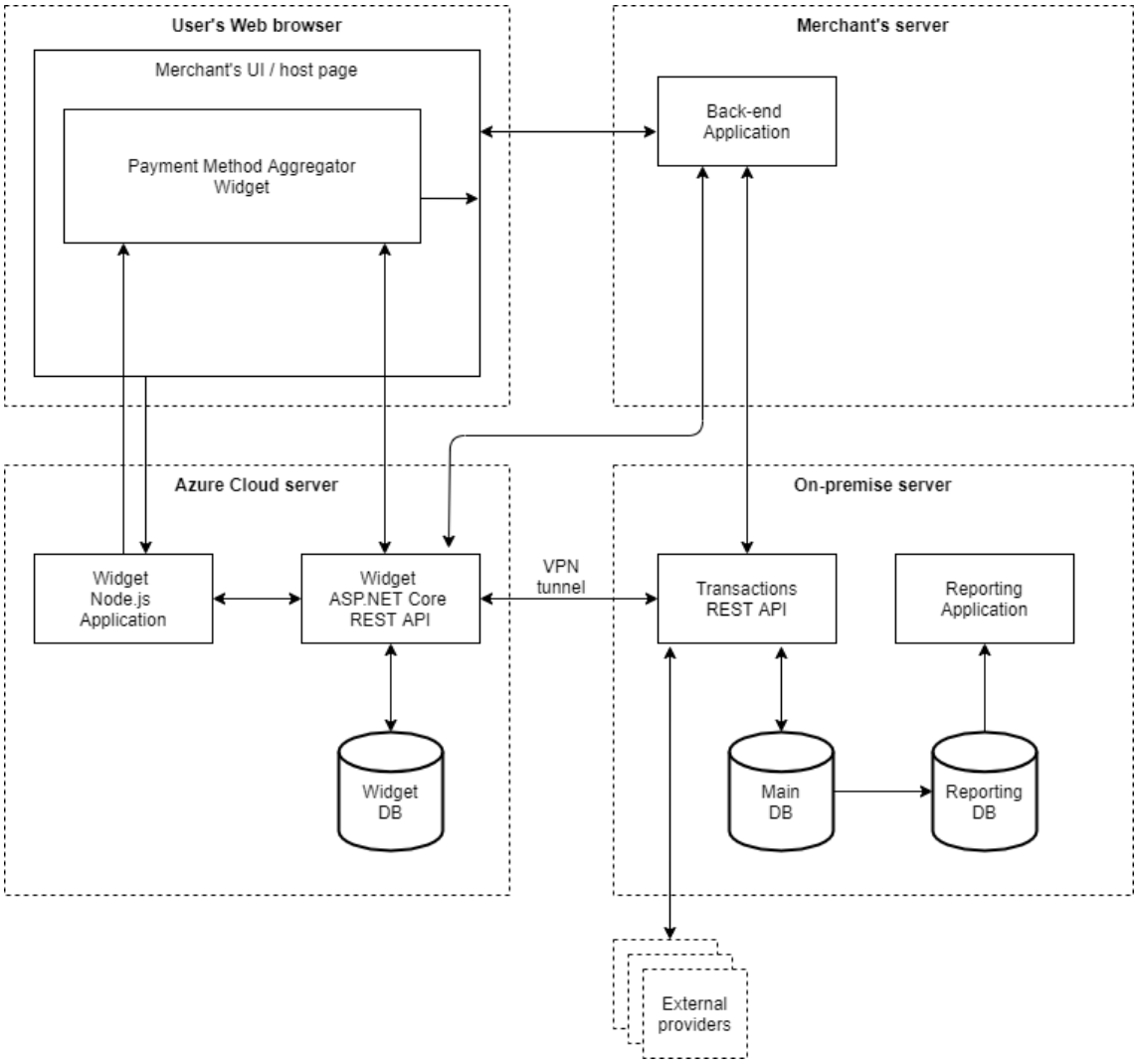


Figure 5. New high-level architecture.

Upon a request from the merchant's web page, the new Node.js application serves a minified widget web page, JavaScript library, stylesheets, icons and other resources. In order to prevent any JavaScript of the host page to access end-user's PCI relevant data like card number or CVC, the payment widget is encapsulated in an iframe preventing access to this data via JavaScript. This enables a shop to bypass PCI DSS requirements as no sensitive payment card data can be accessed by the host page. The form within the iframe will internally handle saving of payment data to the payment aggregator's

database for further processing. To provide an easy integration experience to merchants, the injection of the iframe and the result handling are done by the JavaScript library served by the new Node.js application. The communication from the widget to the host page is implemented based on Web Messaging. A message is posted from the iframe to the parent using a window.postMessage() call with the targetOrigin parameter specified, so that it limits cross-domain messaging to only two involved domains – host page and widget domains.

The purpose of widget ASP.NET Core application is to handle all actions that are directly connected with widget. For example, to trigger widget initialization and save gathered data, and to retrieve basic widget settings.

Each request to the widget back-end ASP.NET Core application must be done using the HTTPS protocol with TLS/SSL, requests must also have Authorization header with the OAuth 2.0 Bearer token specified. The token can be obtained as a separate request to the new widget REST API "token" endpoint with the username and the password. Another possibility is to get the token with the first server-to-server request including the username and the password to the request. Once the token is expired the merchant can refresh it or obtain a new one.

The following figures show an example of separate OAuth 2.0 bearer token request (Figure 6) and response (Figure 7), and the usage of Authorization header with the token for making further API requests (Figure 8).

```
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Og==
User-Agent: PostmanRuntime/7.26.10
Accept: */*
Cache-Control: no-cache
Postman-Token: ad5519cf-cdc3-4eb1-97bf-31361aaba2f1
Host: localhost
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 73

grant_type=password&username=test_vladimir&password=Test
_vladimir1&scope=
```

Figure 6. OAuth 2.0 token request.

```
{
      "access_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJNSSI6IjEiLCJ1c2V
ybmFtZSI6InRlc3RfdmxhZGltaXIiLCJIb3N0IjoibG9jYWxob3N0Iiwi
aXNzIjoiaHR0cHM6Ly9sb2NhbGhvc3QvUG1nLkhvc3QuUmVzdEFwaSIsI
mF1ZCI6IlBtZy5Ib3N0LlJlc3RBcGkiLCJleHAiOjE2MTkyOTM3MDcsIm
5iZiI6MTYxOTI5MDEwN30.Ub5zOTzXHDh-
pzaawXLDLAAo7Ba31oCInhM8nvEOTbU",
      "token_type": "bearer",
      "expires_in": 3599,
      "refresh_token":
"m2wHcnQfXYmeWb_4b4D7QlfQetnKKuTw1PjTHWWpMKXI-
6_eOvbg04U1OFP90hxYB4fBBWpBdIh-BvpUTDwwBrAlMXd-
08LhGwSitYjXoWnL8-
65bN1QLCicjUtsEYsI8ONtZ0o0lvtD0QyR5SQQTotQYxQgcrULyWwSW3Q
E7awxgl9oC3AUNM3WC6n5Pl8qWnWH3hShHwXUhfjPp__rBjD8Fz_8IqyZ
SVgtyruqb5ZY_e8ADkCAWRBweZyqdDJpbkxnBA"
}
```


Figure 7. OAuth 2.0 token response.


```
Content-Type: application/json
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJNSSI6IjEiLCJ1c2Vy
bmFtZSI6InRlc3RfdmxhZGltaXIiLCJIb3N0IjoibG9jYWxob3N0Iiwia
XNzIjoiaHR0cHM6Ly9sb2NhbGhvc3QvUG1nLkhvc3QuUmVzdEFwaSIsIm
F1ZCI6IlBtZy5Ib3N0LlJlc3RBcGkiLCJleHAiOjE2MTkyOTM3MDcsIm5
iZiI6MTYxOTI5MDEwN30.Ub5zOTzXHDh-
pzaawXLDLAAo7Ba31oCInhM8nvEOTbU
User-Agent: PostmanRuntime/7.26.10
Accept: */*
Cache-Control: no-cache
Postman-Token: 94b3a003-531a-4657-88e8-a486901a59f5
Host: localhost
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 39
```


Figure 8. Example of Bearer token usage.

The application flow diagram reflected in Figure 9 shows simplified the 4 stages of the whole payment aggregator process. In the context of this thesis the first 3 stages are covered and prototyped as they reflect the flow of the new web application.

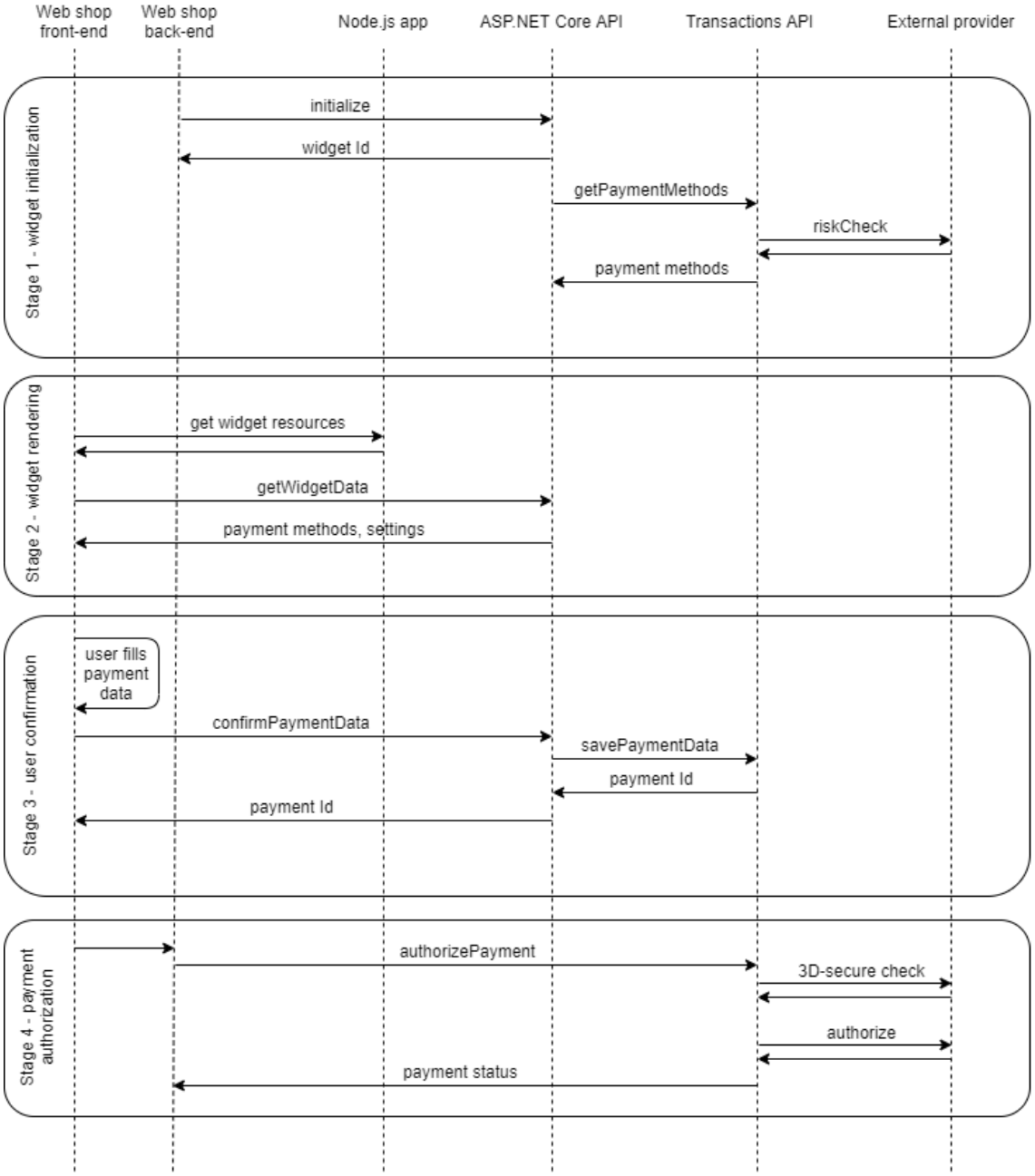


Figure 9. Application flow diagram.

5.2 Widget initialization

Widget initialization is the process of retrieving merchant settings and generation of available payment methods that will be offered to the shopper. When this is done, the payment methods data is stored in the Widget database and is available for fast widget rendering once needed. Initialization of the widget must be done prior to rendering the widget page. The merchant can call initialization whenever it is reasonable to initiate

data preparation. The good time for this could be the moment when shopper logges into the web store or adds the first item to his shopping cart.

Available payment methods depend on several factors: merchant settings, store and shopper geographical location, shopper credit behaviour and risk profile, and many more. In case merchant would like to offer a possibility for payment slicing, the user's risk profile must be evaluated at an external provider, also a query to partly payments provider should be executed in order to get possible options based on the shopper's personal data, shopping cart and risk profile. All these operations and external calls may take time depending on the number of external calls and availability of third-party services. That is why it is so important to trigger initialization beforehand and not on the widget rendering, creating redundant waiting time for the shopper.

Initialization is triggered by the merchant as JSON-formatted server-to-server request to the "initialize" endpoint of the new widget Core application. It may include the desired payment methods, customer data and the widget version, as shown in Figure 10.

```
{
    country:"UK",
    culture:"en-GB",
    paymentMethods:"MasterCard,Visa,Apple
    Pay,PayPal,Sepa",
    version:"1.0"
    billingCustomer:{
        firstName:"Steven",
        lastName:"Smith",
        customerId:"100014",
        email:"smith@email.com",
        dateOfBirth: "1983-02-15",
        debitorNumber: 1000141,
        address:{
            streetName:"Baker Street",
            streetNumber:"5",
            postalCode:"778004",
            city:"London",
            country:"UK"}
    },
    shippingCustomer:{
        firstName:"Steven",
        lastName:"Smith",
        customerId:"100014",
        email:"smith@email.com",
        dateOfBirth: "1983-02-15",
        debitorNumber: 1000141,
        address:{
            streetName:"Baker Street",
            streetNumber:"5",
            postalCode:"778004",
            city:"London",
            country:"UK"}
    }
}
```

Figure 10. Widget initialization request.

Web service retrieves merchant widget settings from the widget database and responds with the data that can be later used for rendering the widget. The response (Figure 11) contains generated widget identifier, initialization URL, successful operation marker, but may also contain access token or any additional resources and custom scripts, that will be used in widget rendering step.

```
{
  "widgetId": "208877be-03b3-42f9-a925-144763bbeb36",
  "success": "True",
  "initalizationUrl":
"https://www.gateway.com/widget/1.0/208877be-03b3-42f9-
a925-144763bbeb36",
  "accessToken": "Bearer <token>",
  "resources":
"https://www.gateway.com/widget/1.0/merchant/resources",
  "scripts":
"https://www.gateway.com/widget/1.0/merchant/scripts.js"
}
```

Figure 11. Widget initialization response.

After the mentioned widget initialization response, the application sends a request to the existing Transactions API in order to proceed with data analysis and payment methods preparation, which typically takes a few seconds but can also take a while. Transaction API responds with JSON-formatted dataset and widget Core API stores data in the WidgetStates table of the widget database under the widget id that was generated before. The structure of WidgetStates table is given in the Table 1.

Table 1. WidgetStates table columns.

Field	**Data type**	**Description**
Id	guid	Widget initialization identifier
MerchantId	int	Merchant identifier
Request	nvarchar[1000]	Widget rendering request
State	nvarchar[max]	JSON data for rendering the widget
PaymentId	int	Saved payment data identifier
UserAgent	nvarchar[max]	User's web browser user-agent
CreateDate	datetime2	Entry creation date and time
UpdateDate	datetime2	Entry update date and time

After data needed for widget rendering is stored into WidgetStates table, and when the shopper would like to pay for items in his cart, the merchant web page should send a request to render the payment widget in iframe inside the host page.

5.3 Widget rendering

Widget rendering is a simple and quick process consisting of the following steps.

The Figure 12 shows the GET request to widget Node.js application to load minified HTML, JavaScript and CSS files in an iframe inside the online store host page.

```
https://init.gateway.com/widget/?r={"clientDomain":"https://www.merchant.com/
","initializationUrl":"https://www.gateway.com/widget/1.0/208877be-03b3-42f9-
a925-
144763bbeb36/","resourcesUrl":"https://www.gateway.com/widget/1.0/merchant/re
sources","paymentServiceUrl":"https://www.gateway.com/payment/1.0/","placeHol
der":"innerElement","frameDimensions":"800x600","onContentResized":"","automa
ticResize":"true","widgetTitle":""}
```

Figure 12. Payment page request example.

The fields listed in the Table 2 can be used for the forementioned request.

Table 2. Payment page request fields.

Field	**Description**	**Optional**
clientDomain	Merchant's domain name	
initializationUrl	Widget initialization endpoint URL	
paymentServiceUrl	Payment service URL for saving payment data	
placeHolder	The host page container element ID where payment page will be rendered	
resourcesUrl	Custom widget resources for specific merchant if any	x
paymentSuccessCallback	JS method that executes after the successful payment data confirmation	x
paymentErrorCallback	JS method that executes after the payment data confirmation returned an error	x
frameDimensions	The payment page frame height and width	x
cssUrl	Custom merchant stylesheet file location	x
renderErrorCallback	JS method that executes after unseccessful widget rendering	x
automaticResize	The content of the widget page will respond to the parent page resize and vice versa	x
widgetTitle	The widget iframe title	x
onContentResized	JS function triggered when the widget content is resized	x

When the HTML page and JavaScript library is loaded into user's Web browser it sends getWidgetData JSON-formatted request to the initializationUrl, as shown in the Figure 13, to widget Core API in order to retrieve available payment methods and settings, that were prepared in the widget initialization step.

```
{
   widgetId:"208877be-03b3-42f9-a925-144763bbeb36"
}
```

Figure 13. Widget getWidgetData request example.

A payment page will be rendered based on the received data, so that shopper can choose the preferred payment method and enter all needed payment and personal data depending on the selected method.

Figure 14 illustrates the new UI design of the payment page in case credit card is selected as a payment method; the card brand is detected automatically.

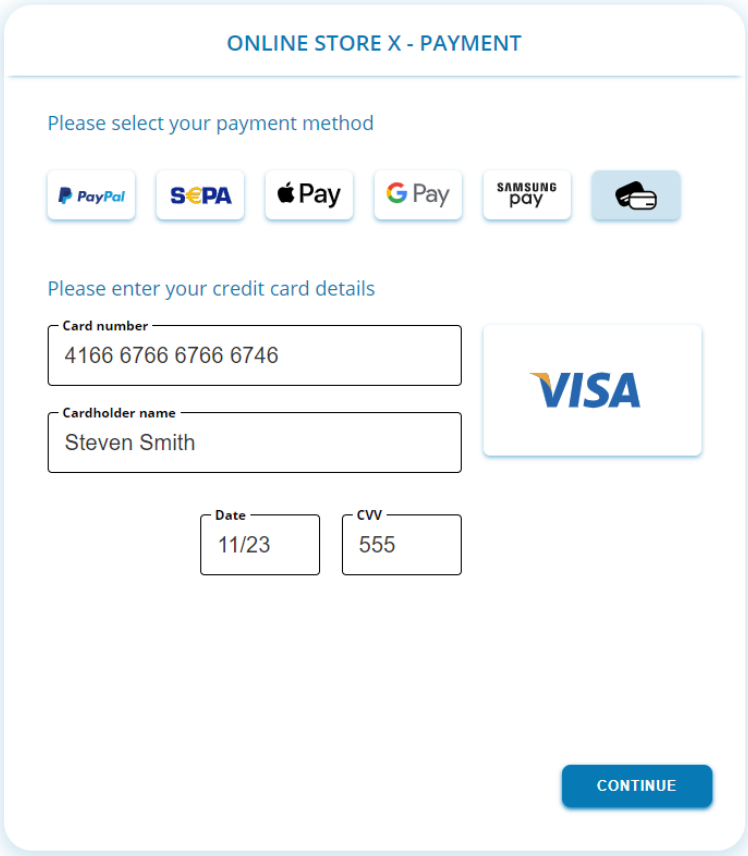


Figure 14. Payment page credit card desktop view.

Figure 15 shows an example of the card number field validation and the error message that informs user about what is actually happened.

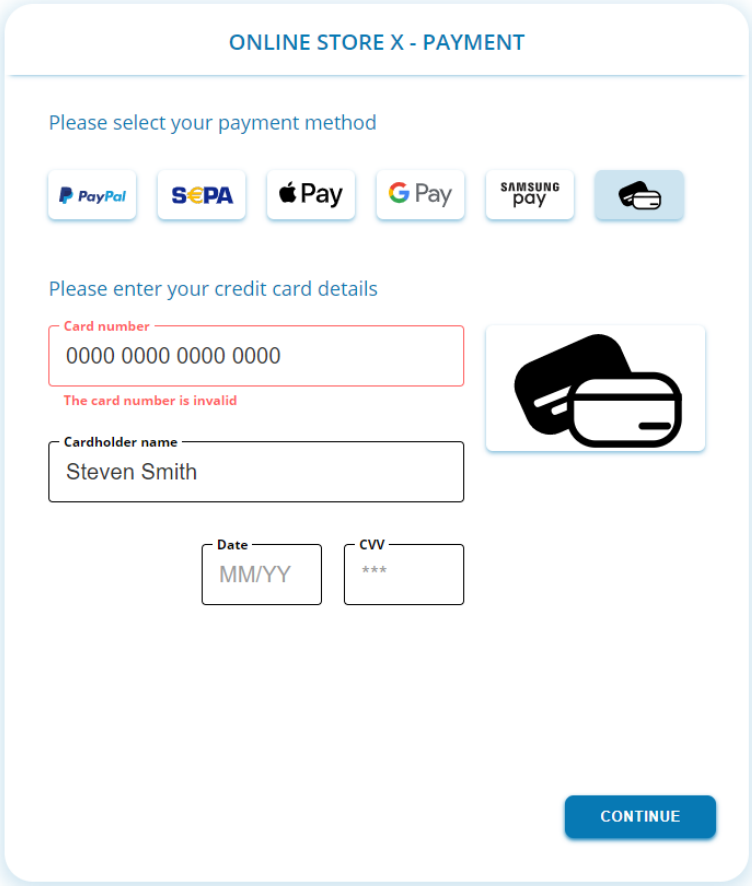


Figure 15. Payment page validation example.

The example of another selected payment method information text is illustrated in the Figure 16.

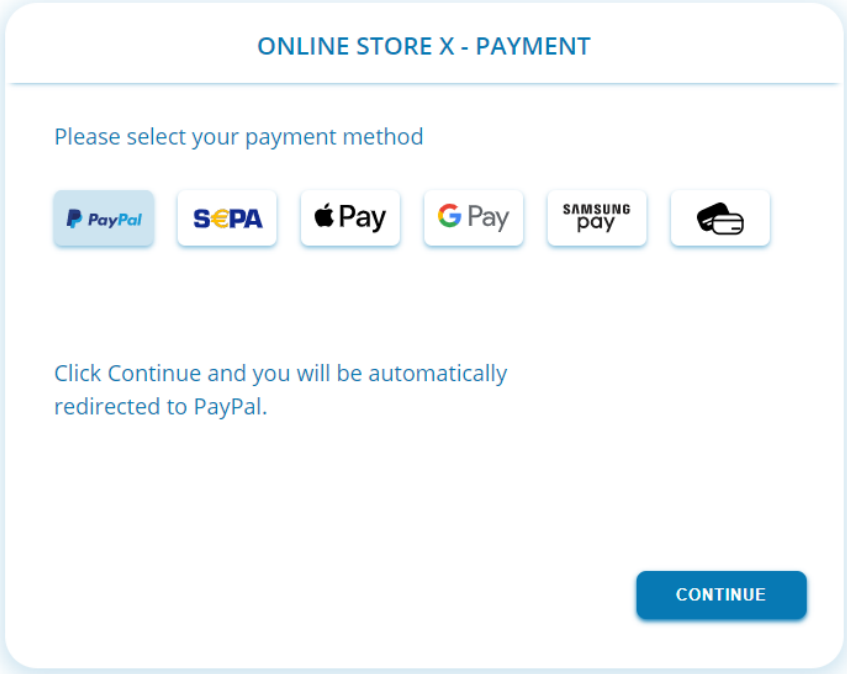


Figure 16. Payment page PayPal desktop view.

Figure 17 displays a mobile view of the payment page – the size and position of buttons, logos and fields are automatically adjusted according to user's screen size.

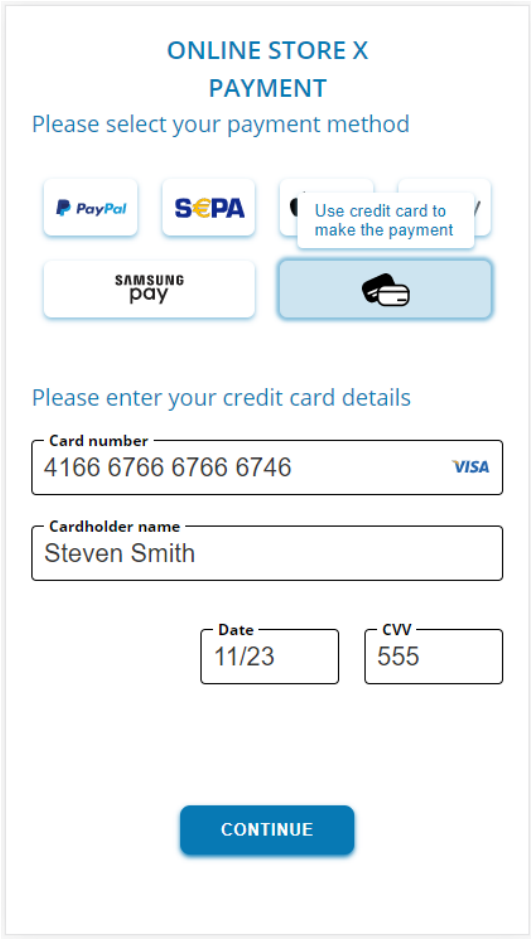


Figure 17. Payment page mobile view and help text example.

5.4 User confirmation

After the shopper has entered payment details and pressed Continue button, the payment page triggers the confirmPaymentData request to widget API which securely stores payment details via Transactions API into the Main database, using RSA encryption. Figure 18 shows the JSON-formatted request example.

```
{
 "paymentMethod":"Visa",
 "paymentDataCard": {
       "cardNumber": "4166676667666746",
       "expirationMonth": "11",
       "expirationYear": "2023",
       "cvc": "555",
       "cardHolder": "Steven Smith" }
}
```

Figure 18. Payment page confirmPaymentData request.

When payment details are stored the merchant can trigger the payment authorization request to Transactions API in order to proceed with the reservation or debiting of the order amount on the shopper's bank account. During authorization Transactions API sends requests to external providers depending on the payment method and payer risk level. In case of need an external provider can initiate 3D Secure check to confirm that payment card is used legally. External verification and payment amount reservation process is not covered by this thesis as the system logic is already implemented in the existing Transactions application.

6 Prototype evaluation

Prototyping allows us to quickly build an initial product design and basic functionality based on ideas and general product requirements. When initially created, software functional prototype should be then evaluated by company customers and product managers. This will provide a highly valuable feedback and help to mitigate the risk of the product development moving in the wrong direction and consuming too much resources.

The prototype developed and overviewed in this diploma thesis was sent for preliminary evaluation to the product manager and 3 key customers of the company.

The evaluation consisted of the following main categories:

- user interface design general assessment;
- usability assessment;
- fields validation testing;
- main functional flow testing;
- payment page performance testing;
- WCAG compliance testing.

Based on evaluators' feedback the payment page user interface design, usability and validation correspond with customers' needs. The UI development can be continued based on agreed product requirements taking stakeholders suggestions in consideration.

Payment page performance testing was conducted based on user interface interaction speed and widget loading time. The Figure 19 shows the Developer Tools output for the widget loading. Multiple tests show the variation of loading time from 105 to 142 ms, but it is always less than maximum possible 200 ms.

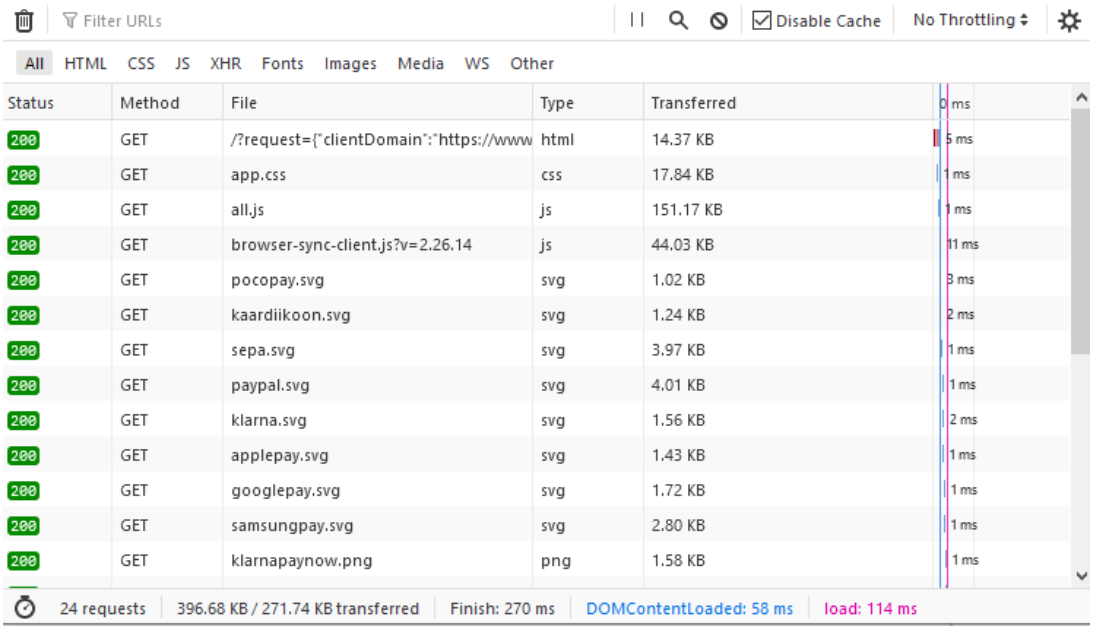


Figure 19. Payment page load speed test.

WCAG compliance test was done using AChecker website located on https://achecker.ca/checker/index.php. The Figure 20 illustrates the results of web accessibility evaluation.

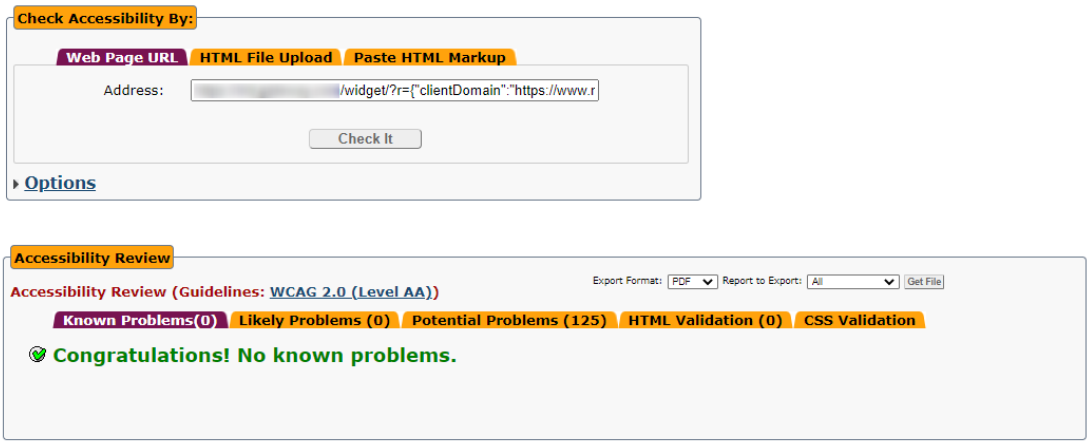


Figure 20. WCAG evaluation results.

The prototype evaluation showed good results from performance and usability perspective, and was approved by management and key customers for further product implementation.

The forementioned evaluation steps must be repeated once prototype will be enhanced or after each product development iteration.

7 Further improvements

Despite the fact that prototype was evaluated successfully, the upcoming product developed on the base of current prototype will need additional enhancements.

Payment page must offer a possibility of scanning the card if shopper is using a mobile device. This will simplify entering the payment card's number and expiry and make overall user experience more interactive. The company should consider using a security-proven third-party library to save development time and use the most up-to-date features.

The card holder's name can be prefilled from the logged in customer shopping cart, so that the payer would not need to enter billing name manually. However, this requirement might need confirmation, that it is legal to prefill any of the user's payment data. Another option is to provide the user a possibility to store all payment credentials in the payment aggregator, so they can be used for the next payment without a need to re-enter them.

Payment page user interface must support translations. The merchant's host page can extract the user's Web browser current culture and pass it along with widget initialization request. Widget Core should be able to retrieve translations either from a static resource, using Azure Functions or third-party translation service. This is essential functionality to be implemented as company customers are located around the globe and obviously shoppers as well.

In order to provide additional security layer, the shopper's payment data can be encrypted using a JavaScript library before transmitting it to the widget API. To implement this kind of encryption, widget Core API should return an RSA public key in an initialization response, so that data can be further encrypted in user's browser. During execution of confirmPaymentData request the payment data would be decrypted server-side using an RSA private key to be able to process the data as needed.

Shopper will not always end up paying for his shopping cart items. Some shoppers will abandon their purchasing process due to various reasons. It would be reasonable to introduce a clean-up process that will weekly delete unrealized widget initialization data from the widget database.

Company might need to develop a new reporting application or database procedure, that will create and export widget initialization reports. For making administration routines easier it is reasonable to implement a new administration application that will provide access to such reports. All reports and logs should not contain any sensitive data like payment card number or CVC.

Idempotency should be implemented into widget ASP.NET Core REST application. This will grant the fact that requested operation will be performed only once even in case of multiple identical requests.

8 Summary

The main goal of this thesis was to gather required information, choose software development technology stack, analyse and create a functional software prototype of the new web application for payment methods aggregator.

The company and the product background were described in the introductory part. The used methodologies, problems and goals were defined before moving to any analytical part of the work. The current high-level architecture and known system deficiencies were figured out and listed in the existing solution description chapter. The next stage of the work was to analyse and provide product requirements for the new web application. Undoubtedly also very important was to analyse most used and performant software development technologies and infrastructure possibilities for smooth development, deployment and running the prototype and software that will be based on it.

The software functional prototype is the result of the work done in scope of the current thesis. The chapter dedicated to prototype creation described the architecture together with security aspects, and specific process stages with examples of information exchange between front-end and server-side applications.

Essential was to evaluate the created prototype to decide about reliability, usability and perspectives of continuing work on the full product. There is a number of possible improvements, so that implementing some of them the second prototype version could be released. The proposed enhancements could be also implemented once the full product development will be started in case the second prototype version would be redundant.

The goal of this diploma thesis was fully achieved, and the outcome is considered successful – the suitable technical solution was chosen, and the functional prototype of web application created. The prototype provides the base for further implementation of all needed business and system logic and development of the full flexible, sustainable and profitable solution. The functional prototype will be used for product development and the fully functional business usable product delivery will be planned according to the company needs and resources.

References

[1] Individuals using the Internet [WWW] https://data.worldbank.org/indicator/IT.NET.USER.ZS?end=2019&start=1990&view=chart&year=2020 (22.04.2021)

[2] United Nations, March 2021, “COVID-19 and e-commerce”, pp 39-43 [Online] https://unctad.org/system/files/official-document/dtlstict2020d13_en_0.pdf (22.04.2021)

[3] JSONP [WWW] https://en.wikipedia.org/wiki/JSONP (22.04.2021)

[4] TIOBE Programming Community Index Definition [WWW] https://www.tiobe.com/tiobe-index/programming-languages-definition (17.04.2021)

[5] Stack Overflow Annual Developer Survey [WWW] https://insights.stackoverflow.com/survey (17.04.2021)

[6] 2020 Developer Survey - Most Popular Technologies [WWW] https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers (17.04.2021)

[7] TIOBE Index for April 2021 [WWW] https://www.tiobe.com/tiobe-index/ (17.04.2021)

[8] Welcome back to C++ [WWW] https://docs.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-160 (17.04.2021)

[9] Stack Overflow – About [WWW] https://stackoverflow.com/company (17.04.2021)

[10] Java (programming language) [WWW] https://en.wikipedia.org/wiki/Java_(programming_language) (17.04.2021)

[11] Best Java Frameworks [WWW] https://www.jrebel.com/blog/best-java-frameworks (17.04.2021)

[12] 2020 Developer Survey - Correlated Technologies [WWW] https://insights.stackoverflow.com/survey/2020#correlated-technologies (17.04.2021)

[13] Spring Cloud Features [WWW] https://spring.io/projects/spring-cloud#overview (17.04.2021)

[14] Spring Cloud Reviews & Product Details [WWW] https://www.g2.com/products/spring-cloud/reviews#reviews (17.04.2021)

[15] What is Python? Executive Summary [WWW] https://www.python.org/doc/essays/blurb (17.04.2021)

[16] The State of Developer Ecosystem 2020 – Python [WWW] https://www.jetbrains.com/lp/devecosystem-2020/python (17.04.2021)

[17] Django introduction [WWW] https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (17.04.2021)

[18] Advantages & Disadvantages of Django [WWW] https://www.codingninjas.com/blog/2020/12/11/advantages-disadvantages-of-django (17.04.2021)

[19] Flask (web framework) [WWW] https://en.wikipedia.org/wiki/Flask_(web_framework) (17.04.2021)

[20] Advantages and disadvantages of using Flask as a web framework [WWW] https://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-Flask-as-a-web-framework (17.04.2021)

[21] C Sharp (programming language) [WWW] https://en.wikipedia.org/wiki/C_Sharp_(programming_language) (17.04.2021)

[22] A tour of the C# language [WWW] https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (17.04.2021)

[23] The State of Developer Ecosystem 2020 – C# [WWW] https://www.jetbrains.com/lp/devecosystem-2020/csharp (17.04.2021)

[24] ASP.NET MVC [WWW] https://en.wikipedia.org/wiki/ASP.NET_MVC (17.04.2021)

[25] ASP.NET overview [WWW] https://docs.microsoft.com/en-us/aspnet/overview (17.04.2021)

[26] Introduction to ASP.NET Core [WWW] https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 (17.04.2021)

[27] .NET Core and .NET 5 Support Policy [WWW] https://dotnet.microsoft.com/platform/support/policy/dotnet-core (18.04.2021)

[28] What's new in .NET 5 [WWW] https://docs.microsoft.com/en-us/dotnet/core/dotnet-five (18.04.2021)

[29] Cloud Computing Market Share [WWW] https://www.t4.ai/industry/cloud-computing-market-share (19.04.2021)

[30] E-commerce in the time of COVID-19 [WWW] https://www.oecd.org/coronavirus/policy-responses/e-commerce-in-the-time-of-covid-19-3a2b78e8 (19.04.2021)

[31] Web Framework Benchmarks [WWW] https://www.techempower.com/benchmarks/#section=data-r20&hw=cl&test=composite (20.04.2021)

[32] Prototype Model [WWW] https://www.javatpoint.com/software-engineering-prototype-model (23.04.2021)

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis

I Vladimir Nitsenko

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis ”Creating a Prototype for the Payment Method Aggregator”, supervised by Toomas Lepikult
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

14.04.2021