УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ
ФАКУЛТЕТ ПО ИНФОРМАЦИОННИ НАУКИ
КАТЕДРА "ИНФОРМАЦИОННИ СИСТЕМИ И ТЕХНОЛОГИИ"

Павел Стоянов Петров

СИГУРНОСТ И ПРОИЗВОДИТЕЛНОСТ ПРИ ДИГИТАЛИЗАЦИЯ НА ФИНАНСОВИ УСЛУГИ

ДИСЕРТАЦИЯ
за присъждане на научна степен "доктор на науките"
по професионално направление
4.6. Информатика и компютърни науки

СОФИЯ
2021

СЪДЪРЖАНИЕ

СПИСЪК НА ИЗПОЛЗВАНИТЕ СЪКРАЩЕНИЯ

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
B2B	Business-to-business
BEAST	Browser Exploit Against SSL/TLS
C2B	Consumer-to-business
C2C	Consumer-to-consumer
CA	Certification Authority
CERN	Conseil Européen pour la Recherche Nucléaire
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
CSV	Comma-separated values
DC	DoubleClick.Net
DNS	Domain Name System
DOM	Document Object Model
DOS	Denial of Service
DRG	Development Research Group
DROWN	Decrypting RSA with Obsolete and Weakened eNcryption
DV	Domain Validation
ECMA	European Computer Manufacturers Association
EV	Extended Validation
FBR	Facebook Retargeting
FTP	File Transfer Protocol
FTPS	FTP over SSL
GA	Google Analytics
GAWCT	Google AdWords Conversion Tracking
GR	Google Remarketing
GTM	Google Tag Manager
HDI	Human Development Index
HES	Hypertext Editing System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPd	HTTP daemon
HTTPS	HTTP over SSL/TLS
IETF	Internet Engineering Task Force
IIS	Internet Information Services

ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
JSON	JavaScript Object Notation
LAN	Local Area Network
NCSA	National Center for Supercomputing Applications
NFS	Network File System
NLS	oN-Line System
OSI	Open Systems Interconnection
OV	Organization Validation
POODLE	Padding Oracle On Downgraded Legacy Encryption
PPPoE	Point-to-Point Protocol over Ethernet
PROMIS	Problem-Oriented Medical Information System
PSD2	Payments Service Directive 2
RFC	Request for Comment
SFNB	Security First Network Bank
SFTP	Secure FTP
SGML	Standard General Markup Language
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TIES	The Interactive Encyclopedia System
TLS	Transport Layer Security
TSV	Tabluation separated values
UN HDRO	United Nations Human Development Report Office
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
W3C	World Wide Web Consortium
WC	Working Group
WHATWG	Web Hypertext Application Technology Working Group
WWW	World Wide Web
XML	eXtensible Markup Language
БВП	Брутен Вътрешен Продукт
БНД	Брутен Национален Доход
ИТ	Информационни Технологии
СУРБД	Система за Управление на Релационни Бази от Данни

УВОД

Дигитализацията на финансовите услуги на настоящия етап се характеризира с висока динамика по отношение на използваните информационни технологии, с разширяване и усложняване на връзката с потребителите, което изисква постоянно усъвършенстване на формите и методите за управление на организациите, предлагащи финансови услуги. Появата на явления като криптовалути, блокчейн транзакции, финтех компании и др. променят пазара на финансови услуги, който традиционно се доминира от банките и други големи финансови организации. За да запазят позициите си в тази силно динамична среда, се налага организациите, предлагащи финансови услуги непрекъснато да внедряват съвременни информационни технологии и да търсят нови подходи за връзка с потребителя на техните услуги чрез дигитализация на своята дейност.

Един от най-важните моменти при изпълнение на тази задача е да се осигури високо ниво на сигурност и на база усъвършенстване и развитие на информационната система, предлагаща финансовите услуги, да се осигури и нейната висока производителност.

На този етап все още съществува липса на достатъчно изследвания в областта на проблемите, свързани със сигурността и производителността при дигитализация на финансови услуги. Успоредно с оперирането в интернет пространството, с дигитализирането на някои финансови услуги, все още банките не прилагат в пълна степен добрите практики по отношение подобряване на сигурността и производителността на своите уеб базирани информационни системи. Особено слабо се отчита значението на използване на сертификати за сигурност от тип OV, при осигуряване на криптирана връзка с потребителите на публично достъпния сайт на банките.

Успоредно с проблемите при осигуряване на сигурността, се

очертават и редица слабости и пропуски при производителността на банковите уеб базирани информационни системи, предлагащи финансови услуги. Може да се отбележи, че все още тези системи не постигат очаквания стопански ефект, сравнено с ново-навлизащите на този пазар финтех субекти.

Периодът след 2010 г. се характеризира с масово навлизане на мобилни устройства от тип смартфон в човешкия бит и ежедневна дейност - те станаха основните уеб клиенти, от гледна точка на клиент-сървър архитектурата. Също така съществени изменения настъпиха и при сървърната част на уеб технологиите - налагането на нови протоколи, сървъри на приложения и формати за съхраняване и обмен на данни в реално време променят не само начина, по който се изграждат информационните системи, но и начина, по който се предлагат финансовите услуги.

За преодоляване на несъответствието във възможностите на информационните технологии и нивото на дигитализация на предлаганите финансови услуги, се извършват интензивни изследвания и разработки от световната наука и практика. Разработването на нови методи и средства за дигитализация изисква изследване на съществуващите практики в сектора на финансовите услуги и дефинирането на нови подходи, към които постепенно да се премине при разработката на информационни системи.

В основата на дигитализацията стои стремежът за преминаване към нови и съвременни форми за организация на бизнеса. Когато за извършване на бизнес процесите преобладаващо се използват електронни средства, то тази организация е известна като "електронен бизнес". Електронният бизнес е форма на организация на бизнеса, при която бизнес процесите, обмяната на информация, финансовите транзакции и прочие дейности се автоматизират чрез информационни системи, които в последните години използват интернет и уеб

технологиите като основна комуникационна среда. Тази среда се използва и за предлагане на различни услуги на мобилни устройства - таблети и смартфони. Постепенно преминаването към тази форма на организация на бизнеса става основна цел на редица компании и банки, появяват се и нов вид финансови компании, наричани с общото наименование "финтех". Дигитализацията на финансовите услуги води до много нови онлайн услуги и по-лесен достъп до банковия пазар за клиентите.

Дори и банки, които първоначално не се занимават непосредствено с електронен бизнес, се намират под неговото влияние, тъй като Интернет и уеб пространството въздействат върху всички аспекти на бизнес дейността. При прилагане на електронен бизнес се появяват възможности за разширяване на базата от клиенти, разширяване на географския обхват, прилагане на електронен маркетинг и др.

Едновременно с новите възможности, които предлага електронният бизнес, се появяват и нови технологични рискове - проблеми със софтуера, базите от данни, идентификацията на потребителите, електронния подпис, конфиденциалността на информацията и др. Преминаването на една банка или фирма към електронен бизнес може да наложи промени в организационната структура и начина на осъществяване на връзка с бизнес партньорите, което от своя страна да окаже влияние върху ролята и задълженията на служителите и мениджърите на различните управленски нива. За минимизиране на рисковете от прилагане на нови информационни технологии е необходимо да се поддържа подходяща информационна инфраструктура и да се следват добрите практики в областта на прилагане на информационните технологии.

Съвременните банкови информационни системи, с които работи крайният клиент, използват главно мобилните и уеб технологии. В

зависимост от нивото на взаимодействие, което използваните уеб технологии предлагат, могат да се разграничат няколко, т.нар. "поколения", уеб технологии:

Първо поколение. Уеб сайтовете представляват подобие на информационно табло, брошура или витрина. Информацията се предава еднопосочно и потребителя не взаимодейства със сайта, а само пасивно възприема информацията, като се придвижва между страниците.

Второ поколение. Уеб сайтът предлага възможност за взаимодействие, чрез система за търсене на определена информация за стока или услуга. Има двустранен обмен на информация под формата на заявка - отговор.

Трето поколение. Уеб сайтът предлага възможност за бизнес взаимодействие - например потребителят може да поръча стоки или услуги и да заплаща онлайн. На сайта има връзки към транспортни фирми, специалисти и други компании, с които компанията има договори.

Четвърто поколение. Уеб сайтът е интегриран в бизнес процесите на компанията и е основно средство за връзка с клиенти и бизнес партньори. Например, поръчката на стока автоматично води до действия по преместването й от склада към отдела за доставка и счетоводното отчитане на операцията.

В последните години голямо развитие се забелязва при системите за електронен бизнес в сферата на търговията (електронна търговия), банковото дело (електронно банкиране), транспорта (например самолетните превози) и туризма (например хотелски резервации). Като основен инструмент за комуникация се наложиха уеб технологиите. Те притежават следните по-съществени предимства:

- Устойчивост. Програмните средства, които се използват, са изпитани в много натоварена среда - Интернет, в продължение на години. Това гарантира надеждност на работата и мащабируемост при

нарастване на интензивността на работа с документите;

- Удобен потребителски интерфейс. Интерфейсните средства, които се използват, са същите, с които потребителите започват да работят с Интернет (много често в домашни условия) и са интуитивно разбираеми. Премахва се или се намалява нуждата от обучение и преквалификация на служителите за работа с хипертекстови документи;

- Универсалност. Използването на стандартизирани приложни програмни интерфейси осигурява независимост от компютърни платформи (например десктоп и смартфони) и операционни системи (например Windows и Android).

В света на финансовите услуги през последните години в световен мащаб се забелязват някои устойчиви тенденции, които оказват силно влияние върху формите на организация на банковата дейност:

Първо, забелязва се увеличаване на индивидуалния подход към клиентите, което е свързано с все по-голямата сегментация на банковия пазар. Това налага да се прилагат нови маркетингови, финансови и технологични концепции, което води до промени и в организацията на банковата дейност в направление на по-голяма гъвкавост и повишаване на синхронизацията с потребностите на клиентите.

Второ, налице е силно и бързо развитие на информационните и комуникационни технологии, което води до повишено търсене на специалисти в областта на информационните технологии за поддържане и развитие на традиционните информационни инфраструктури и информационни системи. Голяма част от дейностите могат да се възложат на външни подизпълнители с цел намаляване на разходите за информационните технологии.

Трето, в резултат на политически промени, обхванали голяма част от държавите по света, се засилва глобализацията. Дава се възможност на бизнес организации и с по-малки размери да навлизат на нови територии и пазари, като за целта се създават териториално

отдалечени структури, които да изпълняват специфични задачи по навлизането на нови пазари и по реализацията на нови проекти.

Четвърто, вследствие на глобализацията се променя бизнес средата на даден пазарен сегмент. Лесното навлизане на нови "външни" конкуренти води до засилване на конкуренцията в конкретния пазарен сегмент. За да защитят позициите си, се налага съществуващите бизнес организации да оптимизират дейността си, като чрез намаляване на оперативните разходи и създаването на нов тип по-гъвкави организационни структури, те биха могли да противодействат по-успешно на конкурентите си.

Горепосочените тенденции са сред основните предпоставки за дигитализация на банковата дейност, която изпитва засилен натиск от страна на финтех компаниите и затова ефективното прилагане на нова организация на бизнеса се оказва ключово за задържане и поддържане на интереса на клиентите към услугите на банката.

Основният фактор, улесняващ възникването на финтех компаниите, е Интернет, който позволява определени финансови дейности да се извършват по-ефективно и с по-ниски разходи. Допълнителен фактор е нуждата на клиентите да ползват високотехнологични услуги, които традиционните банки не предлагат или предлагат на завишени цени. Развитието на мрежовата инфраструктура и особено на т.нар. "последен километър" - там където се осъществява връзката между крайния клиент и доставчика на Интернет, наличието на евтини устройства - смартфони, налагането на социалните медии в бита, са също съществени фактори, които определят възникването и развитието на финтех компаниите.

В редица случаи финтех компаниите обединяват усилията си с банки с цел изпълнение на нов общ бизнес, в който взаимодействието с клиентите да се осъществява основно с електронни средства за комуникация.

Направеният преглед на някои аспекти при дигитализацията в банковата дейност разкрива ролята и значението на уеб технологиите в тези процеси. Не по-малко важни са и проблемите, свързани с повишаване на сигурността при работа с уеб технологиите.

В тази връзка, **целта на дисертационния труд** е да се изследват, от една страна, уеб технологиите като основа за дигитализация на финансовите услуги на банките и в тази връзка да се проучат публично достъпни банкови уеб сайтове по отношение на това какви уеб технологии използват, и от друга страна, какви подходи и инструментални средства са подходящи за увеличаване на производителността на уеб базираните информационни системи, предлагащи финансови услуги. Формулирането на подобна цел е продиктувано от обективния ход на развитие на информационните технологии, измененията в нормативната база (най-вече директивата PSD2 на Европейския съюз и Open Banking във Великобритания) и навлизането на нови финтех организации на финансовия пазар, което изисква адекватни форми за организация на информационната система по отношение на сигурността и производителността.

Постигането на набелязаната цел в дисертационния труд изисква да се разгледат и решат на научна основа две основни групи от **задачи**: методологични и приложни. В първата група се поставят следните задачи:

- Да се изследват и очертаят насоките в развитието на уеб технологиите чрез исторически преглед на най-важните моменти - хипертекст, развитие на основните компоненти на уеб технологиите, софтуерни средства за реализация на уеб технологиите и криптиране на уеб трафика.

- Да се разкрият основните характеристики при технологичното развитие на финансовите услуги и да се разработят въпросите за методологията на управление на финтех организациите и някои

стратегически проблеми във връзка с управление приложението на информационните технологии.

- Да се изследват различните подходи за съхраняване на данни с оглед тяхната ефективна обработка в режим реално време.

- Да се проучат инструменталните средства за съвременна реализация на сървър на приложения (т.нар. "апликейшън сървър") и организацията на работа при предаване на данни при използване на API.

- Да се изследват вариантите за работа на уеб приложенията в режим реално време с оглед подобряване на производителността.

В приложно направление се поставят за решаване следните задачи:

- Изследване на уеб технологиите, прилагани в съвременни публично достъпни банкови уеб сайтове на регионален принцип - балтийски държави, централноевропейски държави и балкански държави.

- Фокусиране на изследването върху използването на средства, повишаващи сигурността на комуникацията при използване на уеб технологии. Във връзка с изпълнението на посочената задача се извършат редица тестове на реално функциониращи банкови уеб сайтове.

- Прилагане на сравнителен анализ между съседни държави с цел установяване на зависимости в практиките на банките в отделните държави по отношение на средствата за сигурност на техните публични уеб сайтове.

- Да се разработят и експериментират алгоритми за оценка на влиянието за повишаване на производителността в следните направления: кеширане на отговор от SQL заявки с NoSQL система, варианти за обработка на данни във формат JSON и времезабавяне (лаг) при използване на различни версии на уеб протокол HTTP.

За **обект на изследване** е избран банковият публичен уеб сайт,

като един от основните компоненти на дигитализиращите се финансови услуги. **Предмет на изследване** са уеб технологиите за сигурност с използване на сертификати и вариантите за подобряване на производителността.

Изследователската теза на труда е, че финансовите институции могат да подобрят, от една страна производителността на информационните си системи като прилагат съвременни инструменти за съхраняване, обработване и обмен на финансови данни, и от друга - да се постигне по-висока степен на сигурност чрез използването на подходящи сертификати. В труда са залегнали някои ограничения във връзка с обхвата на извършените проучвания. На първо място, емпиричните изследвания са ограничени единствено до основните банкови уеб сайтове, без да се проучват допълнителните подсайтове, достъпът до които е ограничен само за индивидуални или корпоративни клиенти. На второ място, с оглед широкият обхват на проблемите, свързани със сигурността, следва да отбележим, че изследването е ограничено единствено до определени информационни технологии, които по наше мнение са основен фактор за подобряване на производителността и в този смисъл други технологии, които също биха могли да доведат до това, не са разгледани.

Теоретична и методологична основа на проведените изследвания са документите от тип RFC, използвани за "де факто" стандартизиране на работата на различни системи и компоненти в Интернет. В качеството на методи на изследването са приложени: системния подход, методът на моделирането, алгоритмизация, проучване и събиране на данни, сравнение, анализ и синтез, систематизиране, индукция и дедукция и научна абстракция.

Основните резултати от изследването по дисертационния труд са публикувани в книги, научни списания и докладвани на научни конференции с международно участие.

ГЛАВА ПЪРВА. ДИГИТАЛИЗАЦИЯ НА ФИНАНСОВИ УСЛУГИ

1.1. Уеб технологиите като база за дигитализация

1.1.1. Развитие на хипертекст и уеб технологиите

При голяма част от бизнес продуктите и услугите в днешно време убедително са се наложили уеб технологиите. Те имат следните по-съществени предимства: устойчивост - програмните средства, които се използват са изпитани в много натоварена среда - Интернет, в продължение на години; удобен потребителски интерфейс - интерфейсните средства, които се използват са същите, с които потребителите започват да работят с Интернет (много често и у дома) и са интуитивно разбираеми, т.е. премахва се или се намалява нуждата от обучение и преквалификация на служителите за работа във виртуална среда; универсалност - използването на стандартизирани приложни програмни интерфейси осигурява независимост от компютърни платформи и операционни системи.

В основата на уеб технологиите стои хипертекст технологията. Нейното значение много често е встрани от полезрението на изследователите, но така или иначе е факт, че развитието на информационните системи, и в частност на уеб технологиите, се обуславя в значителна степен от начина, по който хората ги възприемат не само съзнателно, но и подсъзнателно. Редица психологически изследвания сочат, че начинът на разсъждение при хората много често не е линейно-последователен, а по-скоро прилича на разклонена мрежа (DeBono&Zimbalist, 1970; Vance et al., 2007; Groves&Vance, 2015). В миналото редица учени са правили опити да проектират и създадат системи, които да предоставят информация по начин, различен от линейно-последователния, но можем да твърдим, че по-съществени реални успехи в тази област има през последните 30 г. с масовото навлизане и използване на хипертекст технологията под формата на уеб

страници.

Обхватът и смисълът на понятието "хипертекст технология" може да варира в широки граници в зависимост от контекста, в който се използва. Хипер от гръцки означава над, отвъд, свръх. Терминът хипертекст е предложен от американския учен Теодор Нелсон през 1965 г. (Nelson, 1965).

При хипертекст технологията има възможност текст да се структурира не в линейно-последователна форма, а в йерархична или в мрежова форма, което съществено изменя структурирането и възприемането на информацията. Сред пионерите, които правят опит да приложат "мрежест" подход, може да се посочи испанския архиепископ Изидор Севилски (560–636 г.), който в началото на VII-ми век съставя първата енциклопедия - "Етимология", състояща се от 20 тома (Seville, circa 600; Hispalensis, circa 600). В оригиналния текст към отделните статии след някои думи е имало препратки към други статии и страници, където е можело да се намери повече информация по темата (Miranda, 2006). Това се счита за първообраз на хипертекст връзките, които са в основата на хипертекст технологията и поради тази причина от 1999 г. множество вярващи предлагат на Католическата църква Свети Изидор Севилски да бъде определен за покровител на информационните технологии и Интернет (BBC News, 1999). По-късно папа Йоан Павел II формално го обявява за такъв през 2002 г. (Friedman, 2004; Maignant, 2017). Това показва огромното значение на хипертекст технологията за съвременното общество не само в практически, но и в религиозен аспект.

Хипертекст се използва в редица съвременни приложни програми. Например в недалечното минало голяма част от помощната документация на програмите под Windows беше в хипертекстови файлове с разширение .hlp, .chm и др. подобни. Съществуват и различни десктоп справочни системи като например българските правно-

информационни системи Апис, Сиела, Норма, Дакси, Деюре и др. подобни, които използват подобен подход. В днешно време по-голямата част от помощната документация вече е под формата на уеб страници.

Погледнато в по-широк план, хипертекст технологията няма пряко отношение към създаването, предаването и визуализацията на уеб страници, т.е. тя може да се използва и извън уеб технологиите. Но голямата популярност на хипертекст технологията всъщност се дължи на уеб технологиите, които доминират в Интернет. Към основните компоненти на уеб технологията към настоящия момент може да се причислят: уеб сървър - софтуер за съхраняване и генериране на уеб ресурси; уеб браузър - софтуер за разглеждане на уеб ресурси; спецификации на различни версии на описателни езици HTML, чрез които се създават голяма част от уеб страниците; спецификация CSS - за визуално оформление на уеб страници; спецификация DOM - за реализиране на връзката между уеб страницата от една страна, и CSS и език за програмиране, от друга страна; спецификацията ECMA Script/JavaScript - за динамично генериране на уеб ресурси, проверка на входни данни и създаване на интерактивност; протоколите от приложно ниво на еталонните модели OSI и TCP/IP - HTTP и HTTPS; спецификациите CGI, FastCGI и др. подобни.

Тези компоненти са се наложили като основни. Те са претърпели еволюционно развитие, постоянно се изменят и ще се развиват и в бъдеще. Някои компоненти, реализирани най-често като програмни модули на браузъра - например за Flash приложения и за Java аплети - за създаване на интерактивни документи, постепенно излизат от употреба.

Уеб технологията има връзка и с други компоненти, като хардуер, мрежови устройства, прокси сървъри, DNS-сървъри, операционна система, системи за управление на бази от данни, протоколи от приложно ниво (например SOAP) или протоколи от нива по-ниски от приложното (например TCP, IP, PPPoE) и прочие. Тази

връзка обаче е косвена и можем да твърдим, че уеб технологията до голяма степен е независима от тях, а и те са независими от нея, т.е. възможно е да се използват уеб ресурси и без мрежова връзка, уеб документ може да се разглежда на различни операционни системи, с различен хардуер и т.н.

Уеб страницата в съвременен вариант представлява мултимедийно интерактивно представяне на информация в електронен вид, като чрез хипервръзки е възможно да се свързват различни информационни елементи. Под понятието мултимедиа (multimedia, от англ. multi – много и media – носител, среда) се има предвид среда, интегрираща различни форми за представяне на информация - текст, графика, аудио, видео и прочие форми, които могат да се уловят от човешките сетива. Под понятието интерактивност (от англ. interaction – взаимодействие) се има предвид непосредствен диалогов режим на взаимодействие между човек и компютър. Комбинацията от тези компоненти води до появата на т.нар. явление хипермедия. Под хипермедия се има предвид комбинацията от хипертекст и мултимедия. Терминът хипермедия също е предложен от Теодор Нелсон (Nelson, 1965).

В резултат на развитието на мултимедийните възможности на компютрите, в уеб страниците има възможност да се вмъква мултимедийна информация и в резултат хипертекстовата технология се превръща в хипермедийна технология. Съвременните хипертекстови документи в по-голямата си част са хипермедийни и много често в литературата и в разговорната реч не се прави разграничение между хипертекст и хипермедия, като за удобство се борави само с термина хипертекст.

В исторически план компютърните хипертекст технологии са се развивали съобразно възможностите на изчислителната техника и вижданията на различни изследователи, като в отделни периоди от

време основни системи, базирани на хипертекст технология са били: Hypertext Editing System (HES) създаден в Университета Браун, град Провидънс, САЩ през 1967 г.; oN-Line System (NLS) създаден в Станфордския изследователски институт, град Станфорд, САЩ през 1968 г.; Problem-Oriented Medical Information System (PROMIS) създаден в Университета Върмонт, гр. Бърлингтън, САЩ през 1976 г.; The Interactive Encyclopedia System (HyperTIES) създаден в Университета Мериленд, гр. Колидж Парк, САЩ през 1983 г.; HyperCard създаден от компанията Apple през 1987 г.; Gopher създаден в Университета Минесота, гр. Минеаполис, САЩ през 1991 г. и др. (Conklin, 1987).

В периода 1989-1991 г. започва да се реализира идеята за Световна мрежа (World Wide Web) на Тимъти Джон Бърнърс-Лий (Berners-Lee, 1989; Berners-Lee & Cailliau, 1990) и уеб технологиите се налагат като основна хипертекст технология. В периода 1990-2020 г. уеб технологиите се развиват изключително интензивно, като периодично се включват, изменят и отпадат редица компоненти. Благодарение на налагането на общи, отворени стандарти, голяма част от производителите на софтуер използват уеб технологиите при разработката на информационни системи, при което уеб ресурсите са основно средство за реализиране на потребителския интерфейс. Входните екрани се реализират с помощта на т.нар. форми (чрез HTML таг <form> и тагове на входни полета), а във входно/изходните екрани обикновено се реализират т.нар. навигационни менюта - съвкупност от хипервръзки. С въвеждането на технологията AJAX се дава възможност входните данни да се изпращат асинхронно, като в резултат най-често се променят отделни части от една вече заредена уеб страница. С нея се работи продължителен период от време без да има нужда да се презарежда нова уеб страница.

Най-същественото свойство на уеб страниците е възможността да съдържат хипервръзки - обособени части от документа (част от текст,

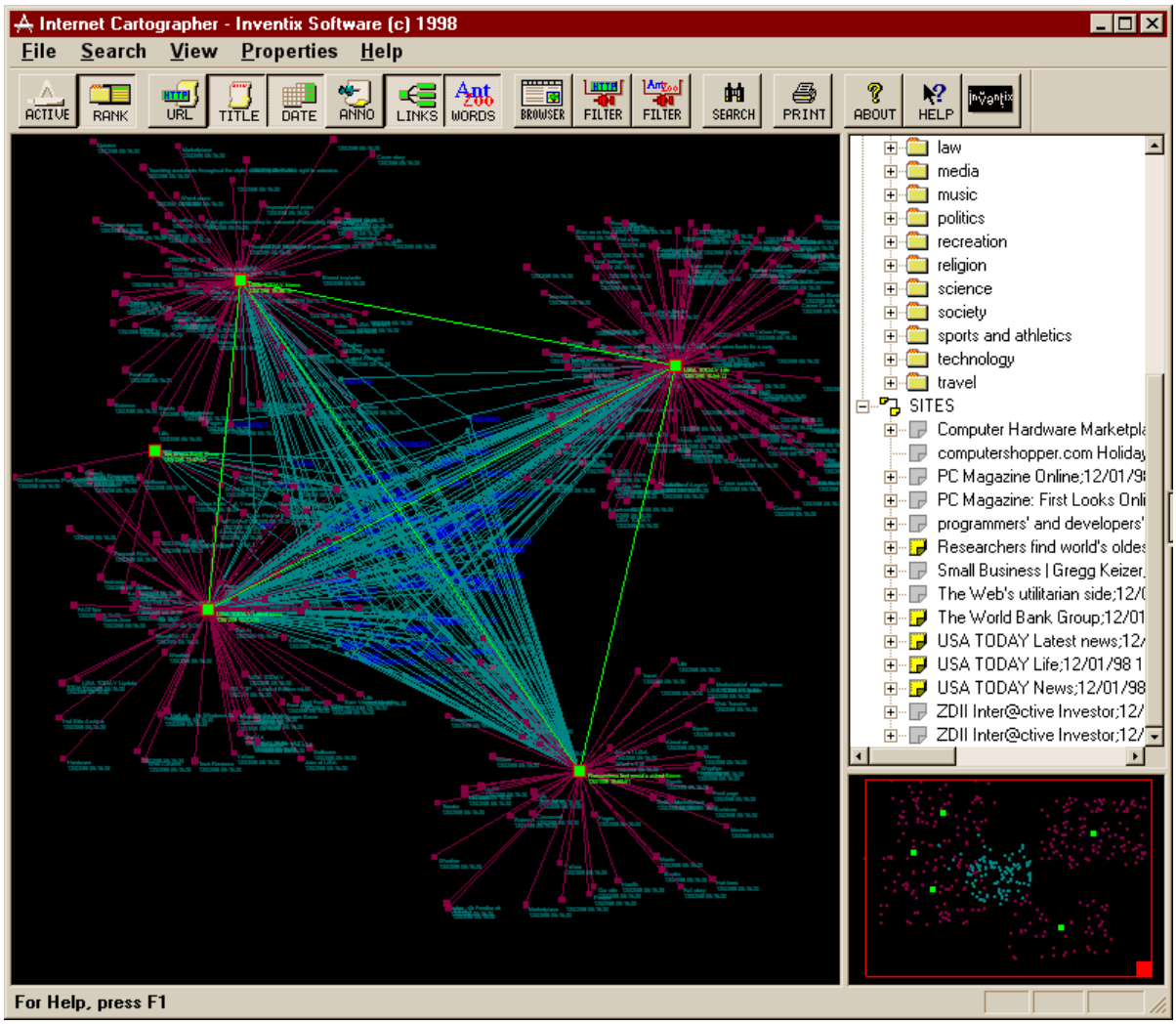
изображение или част от него, софтуерен обект и прочие), които пренасочват към други хипертекстови страници или обикновени файлове, наричани с общото име уеб ресурси. Ключов момент при хипервръзките е системата за адресация на ресурсите, благодарение на която ресурсът може да бъде намерен и получен. В Световната мрежа приетата система за адресация е Uniform Resource Identifier (Berners-Lee, Fielding&Masinter, 2005).

Чрез хипервръзките е възможно да се свържат отделни уеб страници, така че да изглеждат като едно цяло. Свързването с други външни уеб страници позволява на потребителя да получи повече информация по определена тема и увеличава полезността ѝ. Технологията хипертекст е много удобна за потребителите, именно защото лесно може да се премине от една уеб страница към друга. По този начин се получава субективно усещане за непрекъснат и безкраен поток от информация. Чрез следване на определени хипервръзки от отделните уеб страници потребителят може да се фокусира по-бързо върху специфични и интересуващи го теми. Също така, последователността на обхождане на хипервръзките може да се променя в зависимост от предпочитанията на потребителя, а не да се следва линеен модел (Eco, 1996).

От методологична гледна точка, в зависимост от ресурса към който сочат, хипервръзките може да се разделят на две основни категории: навигационни връзки - служат за преминаване към основните части на сайта, и референтни връзки - служат да пояснят или дадат повече информация по даден въпрос (може да се отварят в нов прозорец). Последните споменати връзки могат да бъдат вътрешни - страница от същия сайт, или външни - страница от друг сайт.

С цел да се улесни възприемането на мрежовия характер на хипервръзките, съществуват редица програмни продукти, които визуализират в двумерно или в тримерно пространство семантично

свързани хипервръзки. Обикновено в тези продукти е предвидена възможност за увеличаване на мащаба и интерактивно придвижване в представяното виртуално пространство, както и възможност за филтриране, с цел намаляване на обема на визуалната информация и по-лесно ориентиране (фигура 1.1.1).



Фигура 1.1.1. Екран на Internet Cartographer 2.0 на компанията Inventix Software за визуално семантично представяне на хипервръзки. Източник: http://www.inventixsoftware.com/

Както вече беше посочено, уеб страницата е основен структурообразуващ компонент на уеб технологията. Във визуално отношение тя е ограничена в рамките на един екран на браузъра, който от своя страна може да е с по-големи или по-малки размери от екрана на

монитора на компютъра. При използване на т.нар. фреймове, уеб страницата може да се състои от повече от една уеб страници и най-често чрез AJAX технология те може динамично да се променят в зависимост от действия на потребителя.

В структурно отношение уеб страницата се състои от множество отделни елементи - части от текст (най-често параграфи), графични изображения, програмни обекти - интерактивни области, създавани с различни технологии, които могат да бъдат едновременно и хипервръзки.

Процедурата за създаване и публикуване на уеб страници включва следните основни етапи:

Етап 0. Проучване. Намиране и изследване на други уеб страници по темата с цел запознаване с достиженията в тази област и създаване на интуитивни или формални критерии, които да послужат при вземането на решения на следващите етапи.

Етап 1. Планиране. Основни подетапи:

а) определяне на структура на множеството от уеб страници, изграждащи уеб сайта - разделяне на текста в дадена предметна област на отделни смислови части (теми, статии, раздели). Например за фирмени сайтове е широко прието да има следните раздели: информация за фирмата, предлагани услуги, каталог със стоки, новини за фирмата и информация за контакт. Допълнително може да има и други раздели, в зависимост от дейността. В повечето случаи съдържанието е внимателно съобразено и с демографията на потребителите от целевия пазар.

б) определяне на местоположението и съдържанието на навигационното меню в отделните уеб страници - основни навигационни връзки, с помощта на които потребителят да обхожда отделните уеб страници;

в) разполагане в текстовете на връзки, които да препращат към допълнителна информация или към други теми. Разпределението на

връзките е необходимо да се извършва с цел по-лесно намиране на нужната за потребителя информация.

г) определяне на външния вид (дизайна) - шрифтове, цветови комбинации, графични изображения (иконки), снимки и т.н.

Етап 2. Реализация. Чрез използване на текстов редактор, специализирани уеб редактори, графични редактори и др. приложни програми се кодират уеб страниците съобразно стандартите за уеб технологии. Основните моменти от развитието им е представено на таблица 1.1.1, таблица 1.1.2 и таблица 1.1.3. Възможно е да се използват и готови шаблони на уеб страници. Въпреки наличието на множество помощни програмни средства за създаване на уеб страници, лесно може да се създадат уеб страници с ниско качество. Необходимо е да се обръща внимание не само на това да се раздели текста на части, но и на това да бъде удобно на потребителя да работи с него.

Уеб страниците, за които се предполага, че потребителят може да иска да разпечата, е необходимо да са съобразени с възможностите на принтерния печат и основно да съдържат текстова, а не мултимедийна информация.

Основно внимание следва да се отдели на т.нар. начална страница, с която обикновено започва да работи потребителят, и навигационното меню - съвкупността от връзки за преминаване към основните раздели на сайта, връщане в началната страница и прочие.

3. Публикуване (хостване). В случай, че уеб страниците трябва да са публично достъпни, е необходимо да се качат на хост, свързан с Интернет, на който има инсталирано уеб сървърно приложение, което да изпраща по заявка от браузър уеб страниците и други, свързани с тях, файлове. За целта могат да се ползват редица платени и безплатни услуги по уеб хостинг. Обикновено изпращането (качването) на файловете се извършва чрез протоколи FTP, FTPS, SFTP и HTTP/HTTPS.

4. Поддържане. Периодичното обновяване и актуализиране се извършва с цел да се предотврати остаряването на съдържанието или да се добави нова информация.

1.1.2. Основни компоненти на уеб технологиите и софтуерни средства

За комуникационните протоколи, използвани в Интернет, се прилагат т.нар. "де факто" стандарти, създавани от различни организации по нива на TCP/IP и OSI, като водеща роля има Internet Engineering Task Force (IETF). Стандартите са известни под името RFC (Request for Comment) и първият е създаден през 1969 г.

От таблица 1.1.1, таблица 1.1.2 и таблица 1.1.3 се вижда, че основните стандарти на уеб технологиите се развиват асинхронно, при постоянно разширяване и обогатяване на възможностите. Разработчиците на уеб страници са принудени постоянно да следят последните изменения в стандартите и да се съобразяват с тях, доколкото промените се отнасят за тяхната дейност. Положителен момент е, че при създаване на нови стандарти съществува стремеж те да са обратно съвместими с по-старите стандарти, което дава възможност и "по-стари" уеб страници да се използват на "по-нови" софтуерни платформи. Разбира се, в бъдеще този подход може да се промени, тъй като е възможно някои нови концепции да влязат в неразрешимо противоречие със старите такива, което неминуемо ще доведе до прекъсване на връзката между "поколенията" стандарти.

Таблица 1.1.1. Развитие на основните компоненти на уеб технологиите в периода 1991-2000 г.

	HTML W3C	**CSS W3C**	**DOM W3C**	**JavaScript ECMA**	**HTTP IETF**
1991	HTML Tags (1.0)				HTTP 0.9

1992					
1993					
1994					HTTPS SSL 1.0
1995	HTML 2.0			JavaScript 1.0	HTTPS SSL 2.0
1996		CSS 1		JavaScript 1.2	HTTP/1.0 (RFC 1945) HTTPS SSL 3.0
1997	HTML 3.2			JavaScript 1.3	HTTP/1.1 (RFC 2068)
1998	HTML 4.0	CSS 2	DOM Level 1	ECMA Script 2	
1999	HTML 4.01	CSS 3 проект		JavaScript 1.5	HTTP/1.1 (RFC 2616)
2000	ISO HTML ISO 15445:2000		DOM Level 2	ECMA Script 3	HTTPS (RFC 2818)

Таблица 1.1.2. Развитие на основните компоненти на уеб технологиите в периода 2001-2010 г.

	HTML W3C	**CSS W3C**	**DOM W3C**	**JavaScript ECMA**	**HTTP IETF**
2001	XHTML 1.1				
2002					
2003					
2004			DOM Level 3		
2005		CSS 2.1 проект		JavaScript 1.6	
2006				JavaScript 1.7	HTTPS TLS 1.1
2007					
2008	HTML5 (проект)			JavaScript 1.8	HTTPS TLS 1.2
2009				Node.js	
2010				ECMA Script 5	

Таблица 1.1.3. Развитие на основните компоненти на уеб технологиите в периода 2011-2019 г.

	HTML W3C/ WHATWG	**CSS W3C**	**DOM W3C/ WHATWG**	**JavaScript ECMA**	**HTTP IETF**
2011				ECMA Script 5.1	SPDY (на всички сървъри на Google)
2012		CSS 3 отделни модули			
2013					QUIC (проект на Google)
2014	HTML 5				HTTP/1.1 (RFC 7230)
2015			DOM Level 4	ECMAScript 6 (ES2015)	HTTP/2 (RFC 7540)
2016	HTML 5.1			ECMAScript 7 (ES2016)	
2017	HTML 5.2	CSS 4 отделни модули		ECMAScript 8 (ES2017)	
2018				ECMAScript 9 (ES2015)	HTTPS TLS 1.3
2019	WHATWG living standard		WHATWG living standard	ECMAScript 10 (ES2019)	HTTP/3 "HTTP over QUIC" проект

Освен за създаване на уеб страници, предназначени за публикуване в Световната мрежа, HTML се използва и като основен интерфейс при създаването на приложения, работещи в корпоративни мрежи или за десктоп приложения.

За създаването на уеб страници се използва описателния език HTML (HyperText Markup Language). Езикът първоначално е разработен от Тим Бърнанс-Ли през 1991 г. при работата му в CERN и широко се разпространява заедно с графичния браузър Mosaic, разработен в NCSA. Първоначално HTML е използвал правилата на езика Standard General

Markup Language (SGML), приет от ISO през 1986 г. През 1995 г. се разработва версия HTML 2.0 под ръководството на Internet Engineering Task Force. През 1997 г. работна група на World Wide Web Consortium предлага версия HTML 3.2 в чието разработване са взели участие представители от големи фирми като IBM, Microsoft, Netscape, Novell, Sun и др. Целта на процеса на стандартизация е когато се създават уеб страници, те да изглеждат по един и същ начин в различни браузъри и под различни платформи.

Основен проблем при стандартизацията е наличието на фирми разработчици на браузъри, които владеят голям дял в този пазарен сегмент, при което се опитват да наложат специфични особености. В някои случаи се налага да се разработват различни варианти на една уеб страница, предназначена за различни уеб браузъри. Друг проблем е, че стандартизиращите организации се опитват да удовлетворят все по-голям брой искания за включване на специфични синтактични конструкции, които да обогатят възможностите на уеб страниците. Това води до усложняване на синтаксиса на описателните езици и затруднява тяхното изучаване и използване.

През 1998 г. World Wide Web Consortium (W3C) предлага HTML 4.0, в който се включват възможности като стилове, скриптове, вграждане на обекти и др. В него е приет стандарта ISO/IEC:10646 за множеството от международни символи, които могат да се използват в уеб страниците. Едновременно с добавянето на нови тагове, се прави опит езика да се изчисти от излишните тагове. През 2000 г. ISO приема стандарта ISO/IEC 15445:2000, който се базира на HTML 4.01 и редица създатели на уеб страници се съобразяват с него при своите разработки. През 2014 г. е приет HTML 5, който значително разширява възможностите на езика.

На физическо ниво уеб страницата представлява обикновен текстов файл. Това дава възможност при създаването и редактирането на

уеб страници да се използват както елементарни текстови редактори (например Notepad), така и да се използват специализирани уеб редактори (като MS Front Page, DreamWeaver и др. в миналото), които значително улесняват и ускоряват процеса по създаване на уеб страници.

Форматирането на уеб страницата се извършва чрез специални команди, наречени тагове. Таговете се разполагат между т.нар. ъглови скоби - < и > (символите по-малко и по-голямо). Текстът разположен между ъгловите скоби има специално значение за програмата, която интерпретира уеб страницата - уеб браузърът, и тагът не се визуализира, а се изпълнява като команда. Някои автори предлагат различни начини за анализ, класификация, извличане на съдържание и прочие обработка на база на таговете - последователност, честота на повторение, наличие на съвпадение с определени шаблони и т.н. параметри (Sarhan et al., 2015; Carey&Manic, 2016; Novozhilov et al., 2016).

Графичният дизайн е съществен момент в процеса на проектиране и разработка на една уеб страница: снимане, оформяне на изображения и страници, избор на шрифт, подбор на цветови комбинации. Този процес може да се определи като изкуство и не подлежи на формално описание.

За подобряване на външния вид на уеб страниците се прилагат таблици с каскадни стилове - Cascading Style Sheets (CSS), чрез които се задават различни характеристики, като вид и размер на шрифт, последователност и позиция на извеждане на отделните части на документа и прочие. Под стил се разбира набор от визуални и други характеристики, задавани като стойности на свойства, които могат да се прилагат към HTML и други тагове. Основната идея е да се получи разделяне между съдържанието на документа и неговата логическа структура, от една страна, и физическото визуално оформление, от друга. В резултат от прилагането на CSS, някои учени предложиха идеята за нов вид виртуална мрежа - т.нар. Семантична мрежа, която да

обединява по смисъл различни хипертекстови документи или части от тях (Berners-Lee et al., 2001). Тази идея частично е реализирана.

Към момента има две действащи спецификации на синтаксиса на CSS и една в проектно състояние. С всяка нова спецификация се добавят нови свойства и за изучаването им е необходимо все повече време. Докато в CSS1 има 53 свойства, в CSS2 те са вече 120, а в CSS3 се предвиждат над 250 (работата по стандарта все още не е завършена).

Както се вижда от изложението по-горе, компонентите на хипертекст технологията дават възможност за реализиране на голям брой различни варианти за работа и именно с това разнообразие можем да си обясним високата популярност и разпространение на Световната мрежа, като среда за развитие на хипертекста. На практика прилагането на хипертекстови технологии в среда на Интернет доведе до появата на големи обеми текстова и мултимедийна информация. Това, от една страна, значително увеличава възможностите на потребителите за запознаване с различни теми и въпроси, но и същевременно затруднява намирането на нужната информация.

Основните проблеми при търсене на информация сред множеството уеб страници е свързано както със самата същност на Световната мрежа - нейния децентрализиран характер, така и с големите обеми информация. По редица въпроси има налични стойностни материали, но повечето потребители не са в състояние да ги открият, тъй като неправилно задават ключови думи в специализираните системи за търсене и в резултат списъка с резултатите е прекалено голям или не е релевантен на очакванията, за да се изследва в разумен период от време.

Методът на свободното обхождане ("сърфиране") е подходящ при обхождане на малка лесно обозрима съвкупност от сайтове, но при по-голям брой обхождани сайтове, процеса започва да прилича на обхождане на лабиринт, при което се загубва общата представа.

Въпреки наличието и силната конкуренция между

специализираните сайтове за търсене на информация в Световната мрежа, проблемите не са решени изцяло, а вероятно няма да бъдат решени изцяло и в бъдеще.

Методологията на хипертекст технологията се реализира в практиката чрез различни софтуерни средства. Едни от тях служат за създаването на статични хипертекстови документи, други се използват за да се разширят възможностите им, трети се използват за преглеждане, четвърти за съхраняване, динамично генериране, изпращане по заявка и т.н. В таблица 1.1.4 е систематизирана информация за появата и еволюцията на някои по-съществени програмни продукти, използвани за реализация на хипертекст технологията.

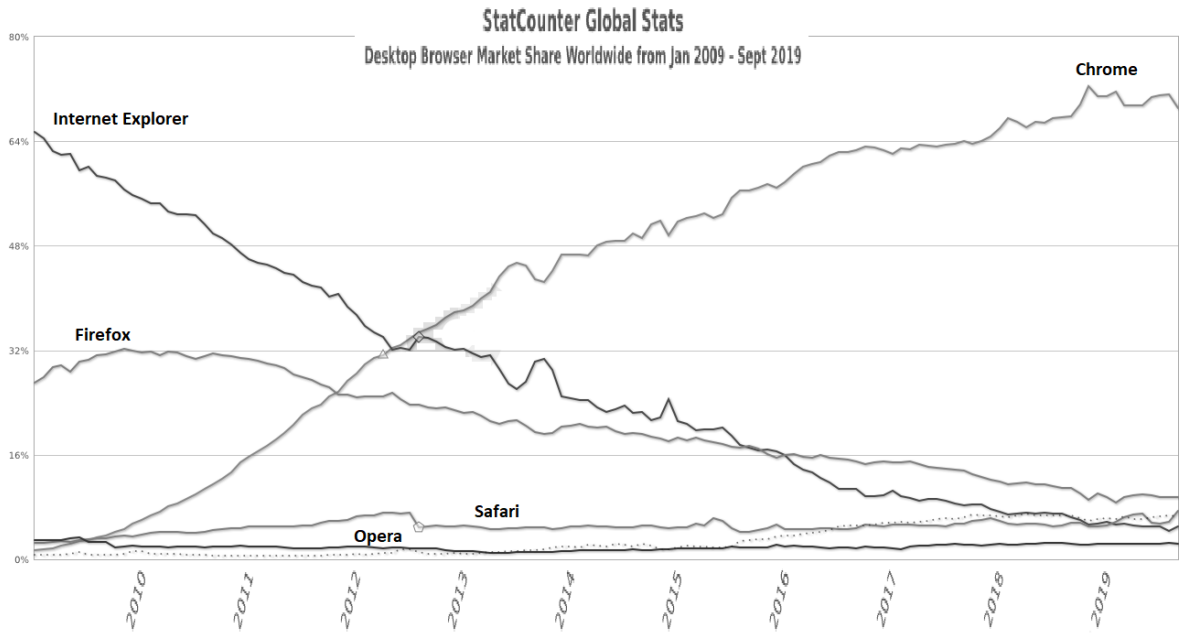
Таблица 1.1.4. Ретроспекция на по-съществените програмни продукти използвани в хипертекст технологията в периода 1991-2009 г.

	уеб браузър	**Flash**	**уеб сървър**
1991	WorldWideWeb		CERN httpd 0.1
1992	Lynx 1.0		
1993	NCSA Mosaic		
1994	Netscape Navigator 1.0		NCSA HTTPd
1995	Internet Explorer 1.0		Apache 1.0
1996	Netscape Navigator 3.0, Internet Explorer 3.0	Macromedia Flash 1	IIS 1.0
1997	Netscape Navigator 4.0, Internet Explorer 4.0	Macromedia Flash 2	IIS 2.0
1998	Netscape Communicator 4.5	Macromedia Flash 3	
1999	Internet Explorer 5.0	Macromedia Flash 4	Apache 1.3
2000	Internet Explorer 5.5	Macromedia Flash 5	IIS 5.0
2001	Internet Explorer 6.0		
2002	Mozilla Netscape 7.0	Macromedia Flash MX	Apache 2.0
2003	Opera 7.0	Macromedia Flash MX 2004	IIS 6.0
2004	Mozilla Firefox 1.0		
2005	Firefox 1.5	Macromedia Flash 8	
2006	Internet Explorer 7.0, Firefox 2.0		

2007	Netscape Navigator 9	Adobe Flash CS3	
2008	Firefox 3.0	Adobe Flash CS4	IIS 7.5
2009	Google Chrome 2.0, Firefox 3.5, Internet Explorer 8.0, Opera 10.0 Gazelle 1.0		Apache 2.2.13

Браузърът е основно средство за разглеждане на хипертекстови документи и до голяма степен оказва влияние при разработката им, тъй като неговите възможности са лимитиращи по отношение качеството на възприемане на информацията. До 2010 г. водещо и монополно положение в този пазарен сегмент има компанията Microsoft с Internet Explorer, а неин основен конкурент е Mozilla с Firefox (фигура 1.1.2). С появата на браузъра Google Chrome ситуацията се променя. Някои разработчици на уеб браузъри, като например Opera, прилагат стратегията "следвай пазарния лидер", като се опитват да създадат максимална съвместимост с Internet Explorer (в близкото минало). Други разработчици правят опити стриктно да спазват стандартите на W3C, но пазарния им дял е много малък. В миналото е съществувало и е възможно отново да се появи явлението "война на браузърите", както това беше в периода 1995-1998 г. между Netscape Navigator/Communicator и Internet Explorer. Индикации за това са тенденциите браузърът, от една от многото програми, инсталирана на съвременните компютри, да се превърне в основния и може би в единствения приложен продукт, който ще е инсталиран в бъдеще. Можем дори да очакваме трансформация на браузъра в самостоятелна операционна система.

Към 2019 г. делът на браузърите (фигура 1.1.2), използвани при десктоп системи е: Chrome - 68,9%, Firefox - 9,3%, Safari - 8,7%, Edge - 4,5%, IE - 4,4%, Opera - 2,3% (Statcounter.com, 2019).



Фигура 1.1.2. Използвани браузъри при десктоп системи в периода 2009 - 2019 г. (Statcounter.com, 2019)

1.1.3. Криптиране на уеб трафик

При използване на текстови протоколи (каквито са HTTP/0.9/1.0/1.1), след прослушване на мрежовия трафик, обменяните заявки и отговори лесно могат да се разчетат, дори и без използване на сложни технически средства. Обмяната на поверителни данни (пароли, номера на банкови карти за разплащане, лична информация и пр.) налага използването на протокол HTTPS, с чиято помощ обменяните данни се криптират. Използването на сигурна връзка увеличава изчислителното натоварване на клиента и сървъра по отношение на заето процесорно време и обем заета оперативна памет, но в последните години това не се счита за сериозен аргумент, в предвид големите ползи, които носи осигуряването на сигурност на връзката.

Протоколът HTTPS, предпазващ от подслушване на трафика, обменян между уеб сървъра и уеб клиента, е известен още и като "HTTP Secure", "Secure HTTP", "HTTP over SSL", "HTTP over TLS", "HTTP over SSL/TLS" и др.). Уеб сайтовете, които използват протокола HTTPS,

според добрите практики, използват сертификати, издавани от т.нар. сертифициращи органи (certification authority, CA), които удостоверяват връзката между публичния ключ, използван при криптиране на връзката и името на домейна, записано в обслужващите сървъри на имена (DNS). Тези сертифициращи органи подписват с методите на електронния подпис сертификат във формат X.509, който свързва публичния ключ, заедно с други мета данни, отнасящи се за притежателя на този ключ (Cooper et al., 2008). В случая на уеб сайтове, най-съществена е информацията за DNS името на сайта. Притежателят на публичния ключ съответно трябва да може да администрира уеб сървъра, обслужващ това име. Уеб браузърите разполагат с вграден списък, състоящ се от доверени сертифициращи органи, и могат да проверят дали един сертификат е издаден от такава организация. Този списък се актуализира периодично при обновяване на софтуера.

Сигурността при използване на HTTPS се постига в няколко различни направления: осигурява се автентификация на сървъра посредством цифрови сертификати, издавани от доверени сертифициращи органи; евентуално сървърът може да извършва автентификация на клиента, ако той също има сертификат; осигурява се високо ниво на поверителност на обменяните данни между двете страни, като данните се криптират; криптирането допълнително осигурява и интегритета на обменяните данни, т.е. гарантира се тяхната цялостност и не е възможно незабелязано да се подменят част от тях.

Например, в началото на комуникацията уеб клиентът, който иска да се свърже със съответния банков уеб сървър, няма как да установи, че наистина се свързва с правилния сървър. В хода на криптиране на комуникационната връзка банковия уеб сървър изпраща публичния ключ, който ще се използва за криптиране на връзката и сертификата. Браузърът проверява сертификата дали не е с изтекъл срок и дали DNS името отговаря на записаното в сертификата. Обикновено

сертифициращият орган не подписва директно сертификатите, а това се извършва от други организации - посредници, които извършват тази дейност. Така се получава верига от сертификати, които в крайна сметка свързват междинните организации със сертифициращ орган и сертификата на уеб сайта.

В практиката се използват три вида сертификати:

- Валидиране на домейн (DV - Domain Validation);
- Валидиране на организация (OV - Organization Validation);
- Разширено валидиране (EV - Extended Validation).

При Валидиране на домейн (DV) сертифициращият орган извършва проверка дали кандидатът може да използва конкретно име на домейн. Не се извършва проверка за фирмена идентичност и съответно не се показва друга информация в браузъра, освен че връзката е защитена. При Валидиране на организация (OV) сертифициращия орган допълнително извършва проучване за организацията, която се показва при разглеждане на сертификата. При Разширено валидиране (EV) сертифициращия орган извършва задълбочена проверка на организацията по отношение на правната форма на съществуване, реален адрес, право да се използва определен домейн и др. при което името на организацията се показва в браузъра заедно с информацията, че връзката е защитена.

Осъществяването на една връзка по HTTPS е по-сложен процес, в сравнение с HTTP и се състои от по-голямо множество от отделни стъпки. Затова използването на комбинацията HTTPS и HTTP/1.0 (или HTTP/0.9) не се препоръчва, тъй като мрежовата връзка ще се затваря след всеки отговор и при следващи заявки отново трябва да се повтаря процесът по т.нар. "ръкостискане" - "three way handshake" (Postel, 1981). Протоколите HTTP/1.1 и HTTP/2 са по-подходящи за използване през HTTPS, тъй като те дават възможност през една отворена мрежова връзка да се изпращат много заявки/отговори и по този начин

забавянето, получило се в резултат на “ръкостискането” става пренебрежимо малко (Петров & Петрова, 2016).

В най-опростен вид осъществяването на сигурна връзка - "ръкостискане", се осъществява по следния начин: клиентът изпраща заявка за сигурна връзка към сървър, като изпраща списък с криптографски протоколи (алгоритми), поддържани от клиента; сървърът обикновено избира най-сигурния протокол, който и той поддържа, и го връща като отговор; допълнително сървърът изпраща своя цифров сертификат, съдържащ публичен ключ; клиентът проверява сертификата като използва своя списък с доверени сертифициращи органи, след което генерира случаен симетричен ключ (парола) и го криптира по вече договорения криптографски протокол, използвайки публичния ключ на сървъра; сървърът декодира съобщението като използва своя частен ключ и извлича генерирания от клиента симетричен ключ; по този начин двете страни разполагат със специално генериран за връзката симетричен ключ (парола), който е известен само на тях и могат да го използват за криптиране на обменяните по между си данни - заявките и отговорите.

Както се вижда от описанието, процесът е синхронен, налага се клиентът и сървърът да се изчакват няколкократно и в зависимост от различни условия продължителността му може да варира в широки граници - примерно от 50 ms до 1-2 s, а обемът на предаваните данни да достига примерно до 5-10 KB. Това може и да е причината една уеб страница да накара браузъра "да замръзне" за няколко секунди и това най-често се случва, ако чрез HTTPS се зареждат множество компоненти и от други сървъри - например рекламни блокове, бутони за харесване и споделяне в социални мрежи, скриптове за следене на посещаемостта, изображения, шрифтове и прочие. В този случай "ръкостискането" трябва да се извърши с всички уеб сървъри.

При договарянето на криптографския протокол съществуват

различни варианти, известни като SSL/TLS (Secure Sockets Layer/Transport Layer Security) протоколи: SSL 2.0 (Hickman, 1995) , SSL 3.0 (Freier, 2011), TLS 1.0 (Dierks&Allen, 1999), TLS 1.1 (Dierks&Rescorla, 2006), TLS 1.2 (Dierks&Rescorla, 2008). Към 2019 г. протоколи SSL 2.0, SSL 3.0 (т.е. всички SSL варианти) и TLS 1.0 се считат за несигурни и съответно не трябва да се използват, съгласно (Turner&Polk, 2011) и (Barnes et al., 2015). Те са уязвими на различни видове атаки и по този начин са се компрометирали от гледна точка на сигурността. Например SSL 2.0 е уязвим за т.нар. атака DROWN, SSL 3.0 - на атака POODLE, а TLS 1.0 - на атака BEAST. За сигурни протоколи се считат TLS 1.1 и TLS 1.2 (Qualys SSL Labs, 2017). За тях няма публично известни проблеми към този момент, но няма гаранция, че те също няма да се компрометират в бъдеще. От 2018 г. в процес на придвижване като стандарт е TLS 1.3 (Rescorla, 2018). Онлайн проверка, за това какви криптографския протоколи използва браузърът, може да се направи онлайн: Qualys SSL Labs, SSL/TLS Capabilities of Your Browser, <https://www.ssllabs.com/ssltest/viewMyClient.html>. Проверка за това какви криптографски протоколи използва даден публично достъпен уеб сървър може да се направи онлайн. Например следните авторитетни организации предлагат услугите: COMODO SSL Analyzer, <https://sslanalyzer.comodoca.com/>; DigiCert Certificate Inspector, <https://www.digicert.com/cert-inspector.htm>; Qualys SSL Labs, SSL Server Test, <https://www.ssllabs.com/ssltest/index.html>.

Както вече казахме, сървърът и евентуално клиентът използват цифрови сертификати, за да "докажат" своята идентичност. Ответната страна проверява дали сертификатът е издаден от доверен сертифициращ орган. Обикновено браузърите се разпространяват с първоначален вграден списък с подобни органи, който в последствие автоматично се обновява. Допълнително потребителят може ръчно да добавя записи в този списък, с което указва, че се доверява и на други

сертифициращи органи.

Проверката за валидност на един сертификат от страна на браузъра, освен проверка дали е издаден от доверен орган, още включва: дали сертификата се използва между датата на активиране и датата на изтичане на срока на сертификата; дали името на домейна, посочено в сертификата, съответства на домейна, към който е направена връзката; дали сертификатът не е бил анулиран; дали сертификатът не е бил добавен в черен списък и др. проверки.

Основните недостатъци на протокола HTTPS са следните:

- страниците, достъпвани чрез него никога не се кешират в междинен кеш, тъй като връзката между браузъра и уеб сървъра е криптирана и съответно съдържанието не може да бъде разпознато и да се кешира.

- някои браузъри не кешират локално съдържание, получено по HTTPS.

- понеже не се препоръчва да се смесва криптирано и некриптирано съдържание в една страница, то редица ресурси - картинки, икони и други спомагателни файлове, също се криптират и по този начин отпада възможността за намаляване на трафика чрез кеширане.

- криптирането и декриптирането изискват допълнителни изчислителни операции както при сървъра, така и при клиента. Докато при клиента обикновено това не е проблем, то при силно натоварените уеб сървъри, обслужващи едновременно множество HTTPS заявки, това може да се окаже проблем.

- възможно е неправилно конфигурирани firewall или прокси система да не позволяват достъп до HTTPS сайтове. В тези случаи най-често целият трафик се контролира - записва или проверява за вируси.

- използването на сертификати увеличава разходите - цената на година за сертификат за един домейн варира в широки граници и може

да започне от 30-40 лв. и да достигне до 300-400 лв. и повече, в зависимост от типа на сертификата.

В последните години големи организации и компании, като Electronic Frontier Foundation, Mozilla, Akamai, Cisco, IdenTrust и др., създадоха сертифициращ орган, който да издава безплатно сертификати (Robinson, 2017; Aertsen et al., 2017; Aas et al., 2019). Тези сертификати засега са с период на валидност от 90 дни. От началото на 2018 г. започнаха да се предлагат и т.нар. "wildcard сертификати", покриващи всички поддомейни на един домейн. Намаляването или отпадането на разходите за издаване на подписани общопризнати сертификати оказва значително влияние към масово преминаване към използване на HTTPS по света, включително и от българските банки, които все още не използват този протокол.

Съвременните уеб страници се различават съществено от тези, съществували по времето, когато се е създавал протоколът HTTP/1.1 (1997 г.). В този период пропускателната способност на компютърните мрежи и най-вече на така наречения фактор "последен километър" или "последна миля" (телефонни модеми предлагащи скорости до 33-56Kb/s) е най-важния лимитиращ фактор за бързото зареждане на една уеб страница. В днешно време това не е така - масово е наличен т.нар. широколентов достъп до Интернет (над 1Mbps и достигащ до 10Gbps) и основен лимитиращ фактор се оказва самата същност на протокола, който в основата си представлява една логически завършена система от правила.

Основният проблем на протоколите HTTP 0.9/1.0/1.1 е, че те са синхронни по своята същност. При тях се изисква клиентът да изчаква сървъра да върне целия отговор, след което да изпрати нова заявка. При този начин на работа, ако сървърът по-бавно генерира някакъв ресурс, то това на практика блокира цялата последваща комуникация, а обикновено за визуализирането на една уеб страница са необходими

множество отделни ресурси - изображения, скриптове, стилове, шрифтове, фреймове и т.н. За преодоляване на този проблем повечето браузъри отварят едновременно няколко мрежови връзки към сървъра, с помощта на които могат да получават няколко ресурса едновременно (например Google Chrome отваря едновременно 6 мрежови връзки). Това ограничение до няколко връзки е на базата на един домейн и поради тази причина в един момент от време се популяризира тактиката част от ресурсите да се разполагат на други домейни с цел да се позволи да се използват повече мрежови връзки - т.нар. подход за "domain sharding" (Zarifis et al., 2016; Goel et al., 2017). Друга тактика е малките файлове от един и същи тип да се обединят в един голям файл, така че да се минимизира изчакването, получаващо се в резултат на множество последователни двойки заявка-отговори. Например малките изображения се обединяват в така наречените спрайтове - "CSS image sprites" (Marszałkowski et al., 2015; Mjelde&Opdahl, 2017). За решаването на подобни проблеми през 2009 г. Google предлага нов протокол, наречен SPDY, който е в основата на протокола HTTP/2 (RFC 7540 от 2015 г.).

Протоколът HTTP/2 се различава съществено от HTTP 0.9/1.0/1.1 и има редица предимства (Zimmermann et al., 2017; Wijnants et al., 2018; Van Der Hooft et al., 2018) - той не е синхронен, а асинхронен; не е текстов, а двоичен; използва само една TCP/IP връзка за домейн, през която се извършва мултиплексиране, т.е. предаване на множество потоци данни едновременно. Последното се реализира като всяка двойка заявка-отговор се асоциира със свой собствен поток и съответно данните, които трябва да се изпратят, се разделят в отделни фреймове. Фреймовете се асоциират с определени потоци и по този начин през една мрежова връзка може да се предават множество потоци от данни. Потоците са относително независими едни от други, така че блокирането на някой отговор или заявка не пречи на работата на

останалите потоци. По този начин множество заявки-отговори могат да бъдат изпращани едновременно и потоците могат да бъдат приоритизирани. Мултиплексирането намалява необходимостта да се разполагат ресурсите на различни домейни (domain sharding) или да се използват т.нар. CSS спрайтове, но не премахва изцяло тази необходимост. Протоколът HTTP/2 поддържа възможност за компресиране на заглавните части. Това не се извършва чрез класическите алгоритми за компресиране на данни, а чрез организационна техника, използването на която осигурява да няма повторно изпращане на полета от заглавните части, които вече са били изпратени (Yamamoto et al., 2017). За целта и клиентът, и сървърът поддържат таблици с изпратените и съответно получените полета в заглавните части, както и техните стойности. При първата двойка заявка-отговор се изпраща пълната заглавна част, а при последващи заявки-отговори дублиращите се полета се пропускат. Икономията на трафик достига средно до 0,5KB при някой заявки-отговори и това се оказва много ефективен механизъм за значително намаляване на размера на заглавната част.

Друга важна особеност на HTTP/2 е възможността сървърът да изпраща ресурси, които не са били изрично заявени от клиента - това е т.нар. техника на избутване (server push). Тези ресурси се кешират от страна на клиента за бъдеща употреба. Сървърът може да започне да изпраща тези ресурси непосредствено след като връзката е била установена, без дори да чака клиентът да изпрати заявка. При този начин на работа е възможно да се увеличи ненужно мрежовия трафик, но за сметка на това цялостното зареждане на уеб страниците се извършва по-бързо.

Въпреки, че стандартът не го изисква, практическата реализация на поддръжката на HTTP/2 в браузърите налага използването му задължително в комбинация с HTTPS.

В банковата система, както е известно, работи високо квалифициран технически персонал, който отговаря за ИТ инфраструктурата. Основен компонент в последната е уеб сайтът на банката, който в последните години се превърна в основно средство за взаимодействие с клиентите. Характерно за банковата система е, че решенията обикновено се вземат на база анализ на добри практики с цел да се намали риска и в повечето случаи липсва недостиг на финансиране, което дава възможности цената да не се разглежда като основен фактор при избора на софтуер.

1.2. Технологично развитие на финансовите услуги - финтех

Финтех компаниите представляват финансово-технологични компании, които в своята дейност комбинират иновативни финансови и технологични подходи. Тяхната цел е да се конкурират с банки и финансови посредници, за да привлекат клиентите им (Schueffel, 2017). Много често финтех компаниите представляват новосъздадени, стартиращи компании, известни като стартъпи.

Съществуват разнообразни бизнес модели на финтех компании, но тяхна обща отличителна черта е способността им да създават иновации при дигиталните финансови услуги (Петров, 2019). Най-общо можем да групираме тези иновации в следните направления:

- парични преводи и разплащания между контрагенти: C2C преводи на пари - трансфери между физически лица; B2B разплащания - трансфери между юридически лица.

- финансиране: C2C потребителско кредитиране; B2B бизнес кредитиране; C2B групово финансиране - краудфъндинг (crowdfunding).

- управление на пари: финансови съвети от ботове; приложения за финансово планиране; социална борсова търговия (трейдинг); автоматична борсова търговия чрез алгоритми; целеви спестявания.

Финтех компаниите се създават от предприемачи - физически

лица или специално определени служители в съществуваща фирма, които са склонни да поемат икономически риск с цел да се получи печалба. Организационните мероприятия по създаването на финтех компания преминат през няколко етапа.

На нулевия етап чрез маркетингови проучвания се установява потребност на пазара от финтех дигитална услуга (Zavolokina et al., 2016; Anand&Mantrala, 2019). Оценява се потенциалния размер на очакваната печалба чрез изчисляване на вероятните приходи и вероятните разходи от дейностите, свързани с осигуряване на дигиталната услуга. От голяма полза е да се разработи във формален вид бизнес-план, за да се прецени доколко реалистични са очакванията. Ако идеята е реализируема и нивата на печалба или ползите, в случай, че пряко не се цели печалба, са удовлетворителни, то може да се премине към създаване на дигиталната услуга. В противен случай проектът е нерентабилен и няма смисъл да се инвестират средства за развойна дейност и реклама. Например финтех компанията АйКарт АД (iCard AD) е проучила през 2018 г. чрез социологическо изследване потребителските навици по отношение на различните методи за разплащане и изискванията към дигиталните алтернативи. Проучването е на тема "Потребителски опит и нагласи към различни финансови и платежни услуги". На тази база от компанията достигат до извода, че е налице добър потенциал за навлизането на продукта "дигитален портфейл" на българския пазар. Отчетени са изискванията на потребителите в три основни направления: премахване на таксите за обслужване и поддръжка, възможност за проследяване на състоянието на сметките (баланс и движение), възможност средствата в дигиталния портфейл да бъдат достъпни и в брой (Гайдаров, 2018).

На първия етап, въз основа на поставената цел, се създава списък с всички основни и поддържащи процедури, които е необходимо да се изпълнят във връзка с нейното постигане. Съставя се списък с наличните ресурси (финанси, трудови ресурси, бизнес контакти и т.н.), техните

възможности и лимити.

На втория етап се създава подробна спецификация, в която всяка процедура се разделя на достатъчно малки подпроцедури, които да подлежат на контролиране и оценяване. Тази спецификация дефинира дейностите, по разработка на дигиталната услуга и нейното поддържане. Много често по технологичната част на дигиталната услуга се извършва постоянен реинженеринг (Vijaya&Venkataraman, 2018) по отношение на потребителски интерфейс, сигурност, хостинг и облачни услуги, т.е. работата на екипа по разработка не приключва с пускането на първата версия на софтуера, осигуряващ дигиталната услуга.

На третия етап за всяка подпроцедура се задава изпълнител, начало и период на изпълнение. Удачно е да се приложи мрежови график на Гант (Sharon&Dori, 2017), с цел да се определи натоварването и разпределението на времето на всеки изпълнител.

На четвъртия етап на всеки изпълнител се предоставя списък с изпълняваните от него подпроцедури, описание на подпроцедурата, начало, период на изпълнение, от кого да получава указания, на кого да дава указания, през какъв период от време ще се съгласува работата и т.н. Едновременно с това се съставя списък с подпроцедурите и отговорните за тяхното изпълнение, който ще се използва, за да контролира изпълнението на дейностите. При контрола, се отделя специално внимание на специфичните, от критична важност за изпълнение на процедурите, ресурси. За тези подпроцедури, за които няма изпълнители, е необходимо да се търсят допълнително изпълнители, които да се включат в проекта по създаване на дигитална услуга (Titu et al., 2018).

Едно от предимствата на финтех компаниите спрямо банките е, че значително е намален размера на първоначалните средства за започване на нов бизнес. За по-голяма част от ресурсите се заплаща в процеса на изпълнение на работата, т.е. изискването за голям

първоначален капитал отсъства.

Сред първите финтех компании е първата интернет банка, чиято дейност се извършва изцяло онлайн - Security First Network Bank (SFNB) основана през 1995 г. в САЩ (Clark&Lee, 1998). Банката преустановява дейност през 2002 г. Първоначалният вид на сайта на SFNB може да се види на адрес <http://web.archive.org/web/19961114220302/http://www.sfnb.com/>. Успешна компания, която функционира и в днешно време е PayPal, която се създава след сливането на компанията за разработка на софтуер в областта на сигурността Confinity и компанията за онлайн банкиране X.com през 2001 г. През 2002 г. акциите на PayPal излизат на борсата и компанията е изкупена от компанията eBay Inc. (González, 2003).

Финтех компаниите поставят качествено нови изисквания към мениджърите, които управляват подобни бизнес структури (Petrov & Valov, 2019a; Petrov & Valov, 2019b). Освен знания и опит в сферата на финансовите инструменти, са необходими и задълбочени познания в сферата на информационните технологии. Едни от основните задачи на мениджъра са да поставя групови и индивидуални цели, да сплотява и мотивира служителите, и др. За да изпълнява мениджърските си задължения в динамичната среда, в която оперира финтех компанията, на мениджъра са му нужни умения не само да работи и използва информационните технологии, но и да знае как да ги прилага за постигане на определени цели (Hernández et al., 2018). Появява се нуждата от нов тип мениджър, който да прилага специфичен подход за управление във финтех компанията. Без добри познания по информационни технологии и за правилата в света на финансите, мениджърът няма да може правилно да отчита влиянието на технологиите и да взема оптимални решения за управление на финтех компанията. Затова за успешна работа на мениджъра на финтех компанията, освен технически познания, също от толкова голямо

значение са и организационните му умения.

От страна на служителите също се появяват редица нови моменти. В съвременните компании от сектора на информационните технологии е прието част от служителите да работят дистанционно, без точно определено физическо работно място. Липсата на директен контакт намалява рутинните взаимоотношения между членовете на работния екип и нараства значението на самодисциплината, отговорността и професионализма на отделния служител. В случай, че служителят няма точно определено работно място и работно време, би следвало да се има в предвид, че не всички хора са способни да работят успешно в една неформална среда (в домашни условия, на път, сред природата), каквато работна среда реално би могла да съществува в една финтех компания. Способности за самодисциплина и самоконтрол са един от факторите за успеха на един служител в тези случаи (Elshaiekh et al., 2018), защото някои служители започват да се чувстват изолирани и това намалява работоспособността им.

Мениджърите е необходимо да помагат на служителите си да се приспособяват към начина на работа във финтех компанията в случай на дистанционна работа. Честотата на телеконференциите на работния екип би следвало да е такава, че да се избегнат отрицателните ефекти от липсата на директен човешки контакт "лице в лице". Според някои психолози над 50% от информацията, обменяна при директно общуване "лице в лице", е под невербална форма - жестове, мимики, пози и прочие. При общуване само с ел. поща, текстов чат или телефонен разговор тази невербална информация отпада. "Лицето и контакта с очите са сериозен източник на невербална информация и имат огромно значение в процеса на общуването." (Стоянов, 2005, с.118-119). Затова видео връзката би следвало да се предпочита пред другите форми на комуникация.

По отношение на дистанционната работа във финтех компаниите,

мениджмънтът, като съвкупност от принципи и методи на работа при формирането, използването и развитието на персонала, може да се определи, до голяма степен, като изкуство. Ефективното дистанционно управление и координация в случай на множество териториално отдалечени служители е един все още не достатъчно изследван процес. Например, много трудно може да се предположи в кои случаи предоставянето на повече свобода и възможност за самоорганизация на някой служител във финтех компанията ще доведе до положителен ефект или обратното - до лоши резултати. От гледна точка на мениджъра, в повечето случаи, дистанционно работещия служител е като една "черна кутия" - има сведения какво се подава на входа и какво се получава на изхода, но самият процес на работа остава скрит за него и той няма как да въздейства, за да го оптимизира.

На база посочените до момента източници и имайки предвид основните положения, залегнали в теорията на мениджмънта, можем да обобщим, че сред основните задачи, които е необходимо да изпълнява мениджърът на финтех компанията, се включват:

1. Ясно да формулира цели, задачи, срокове и критерии, въз основа на които ще се оценява резултата от работа на всеки служител или приносът му към крайният резултат. Тъй като при дистанционна работа във финтех компания дейността може да се извършва на различни места, на различни часови пояси, по различно време, в зависимост от служителите, то съвместната работа се разтегля в хода на денонощието, и е необходимо да се синхронизира периодически. Възможно е да се стигне до ситуация, при която служител в един часови пояс да не извърши зададената работа в срок, при което се блокира работата на други служители. Затова ролята на мениджъра е много важна - чрез съгласуване на възможностите на екипа да се постигне максимална производителност, което може да се реализира чрез минимизиране на времето за изчакване между служителите и

премахване на "тесните места" в съвместната работа, т.е. с оптимизиране на етапите, които предизвикват верижно блокиране на работата.

2. Регламентиране на минималния период от време между две комуникации за съвместно координиране на работата. Периодичната комуникацията между служителите е от първостепенна важност. Текстовата информация, която се обменя би следвало да е кратка и ясна. Тъй като писмената форма не е подходяща за дълги обяснения, е по-добре да се използва аудио или видео връзка, в случай, че се налагат по-подробни указания. Преди началото на голям проект е добре всички служители да се съберат заедно "на живо" или чрез видео връзка, тъй като практиката е доказала, че трудно се работи в екип, чиито членове слабо или изобщо не се познават. Такива служители не са склонни да са толерантни към грешките на останалите, тъй като средата не го предразполага. Дори нещо повече, колкото повече служителите комуникират помежду си и научават за колегите си (личен живот, хоби, интереси и т.н.), толкова повече се улесняват и служебните контакти помежду им, увеличава се взаимното чувство за доверие, и се подобрява съвместната работа.

3. Създаване и поддържане на чувство за доверие и добра работна атмосфера във финтех компанията. Социалната изолация при дистанционна работа е един съществен проблем, тъй като липсват допълнителните извънслужебни контакти, които са характерни за традиционните организации. Мениджърите следва да имат в предвид и някой особености при дистанционно общуване чрез онлайн средства - например, много често липсата на отговор, мълчанието, демонстрацията на липса на интерес по даден въпрос по-ясно показва отрицателно отношение, отколкото откритото критикуване.

4. Оценяване работата на служителите. От самото начало на започване на дейността по нов проект във финтех компанията, е необходимо мениджърът не само да координира дейността на

дистанционно работещите служители, но той би следвало и да контролира и оценява доколко всеки член изпълнява поставените му задачи. Оценяването е необходимо да се извършва по постигнатия краен резултат, а не по самия процес на работа и поведение на служителя, т.е. факта дали служителят си е на работното място и спазва ли трудовата дисциплина би следвало да е с по-малко значение, спрямо крайния резултат от работата му.

Както и в другите компании, успехът на работа на финтех компанията до голяма степен се определя от човешкия фактор (Haour, 2019) - мотивация, интелектуален капацитет, професионализъм, работни навици и др. Затова към служителите във финтех компанията е добре да се поставят повишени изисквания, като например умения да намират нестандартни решения, широка обща, финансова и техническа култура, добро владеене на средствата за писмена и електронна комуникация, умение за самостоятелно разпределяне на работното време, високо чувство за отговорност към резултата от работата, владеене на чужди езици и др. Служителите, работещи дистанционно, като минимум е необходимо да притежават умения ефективно да комуникират чрез ел. поща, да са запознати с опасностите от вируси и троянски коне, както и за начините за предпазване от тях, да могат сами да инсталират и обновяват нужния им софтуер, редовно да дублират и архивират важната информация, да могат сами да проверяват състоянието на персоналния си компютър, както и да възстановяват данни и програми в случай на инциденти.

Основният фактор, който улеснява възникването на финтех компаниите е Интернет, който позволява определени дейности да се извършват по-ефективно, с ниски разходи. Допълнителен фактор е нуждата на клиентите да ползват високотехнологични услуги, които традиционните банки не предлагат или предлагат на завишени цени, като например операции с криптовалути, онлайн разплащания от тип

C2C, автоматизирана търговия на борси и др. (Karakas&Stamegna, 2018). Развитието на мрежовата инфраструктура и особено на т.нар. "последен километър" - там където се осъществява връзката между крайния клиент и доставчика на Интернет, наличието на евтини устройства - смартфони, налагането на социалните медии в бита са също съществени фактори, които определят възникването и развитието на финтех компаниите.

В редица случаи финтех компаниите обединяват усилията си с банки с цел изпълнение на нов общ бизнес (Romānova&Kudinska, 2016), в който взаимодействието с клиентите да се осъществява основно с електронни средства за комуникация. Пример за подобно сътрудничество е българската финтех компания АйКарт АД (iCard AD) и малтийската банка SataBank Plc.

В някои случаи е подходящо правно да бъде обособено отделно юридическо лице, в което на динамичен принцип по определени проекти, свързани с дигитализация, да работят служители и от двете компании. Тази организация на съвместната работа позволява всеки от бизнес партньорите - традиционната банка и финтех компанията, динамично да преразпределя своите ресурси между своя кръг от специфични задачи и бързо да се реагира при промяна в обстановката.

Основен недостатък на директното сътрудничество е, че появата на временни или по-постоянни проблеми при единия от партньорите може да окаже негативно имиджово влияние на другия партньор (Megaw&Crow, 2018).

Разделянето на съвместната бизнес дейност на отделни части - за банката и за финтех компанията, увеличава разделението на труда, като всеки от партньорите се концентрира в една област, като има възможност за бърза смяна или увеличаване на усилията в случай на необходимост. Тази организация на работа може да се определи като успешна, ако с по-малки разходи, за по-кратко време или с по-голямо качество се изпълняват дейностите по проекти, свързани с

дигитализация, спрямо друга организационна форма.

Много често се разчита на синергетичния ефект, който води до повишаване на ефективността на работа на всеки бизнес партньор и на работата по общия проект като цяло. Банката и финтех компанията обикновено разполагат с някакъв уникален ресурс. Тези ресурси могат да бъдат интелектуални възможности (например при разработка на софтуер или създаване на дизайн), достъп до крайни клиенти (например клонова мрежа или данни от маркетингови изследвания), наличие на лицензи, които дават правото за извършване на определена дейност на определена територия и др.

Уникалните ресурси може да се споделят с общата компания на база на договори с цел да се подпомага дейността. От своя страна всеки ресурс може да се използва и за други проекти, по различно време и така уникалните ресурси да се споделят динамично, в зависимост от моментните потребности на двете бизнес организации. От тази гледна точка съвместната компания има временен характер (Wallace, 2001), спрямо продължителността на живот на всеки от бизнес партньорите. При траен ефект от взаимното сътрудничество, общата компания може да просъществува и дълги, неограничени по продължителност, периоди от време.

В литературата са описани и систематизирани множество варианти за сътрудничество между банки и финтех компании (Drasch et al., 2018). От гледна точка на водещата роля и продължителност на съществуване на съвместната дейност между банка и финтех компания, според нас, формите на сътрудничество могат да категоризират в следните четири основни варианта:

- Дългосрочно сътрудничество с водеща банка. Банката предоставя част от своите спомагателни дейности на съвместната или директно на финтех компанията на база на дългосрочни договори. Например: разработка и поддръжка на банков софтуер, облачни услуги,

онлайн чат обслужване на клиенти при технически проблеми, разработка на интерфейс, проектиране на база от данни, кодиране, тестване и др.;

- Краткосрочно сътрудничество с водеща банка. Краткотрайна съвместна работа с точно дефинирана и достижима цел, в която водеща роля има банката - най-често да се отговори бързо на някаква специфична потребност на пазара на дигитални банкови услуги.

- Дългосрочно сътрудничество с водеща финтех компания. На базата на дългосрочни договори, общата или финтех компанията използва ресурсите на банката, за да развива своята дейност. Например: стъпване на нов, силно регулиран пазар; използване на клоновата мрежа за предлагане на нова услуга за клиентите; използване на банката като трезор и депозитар на финансовите ресурси, с които оперира финтех компанията.

- Краткосрочно сътрудничество с водеща финтех компания. Краткотрайна съвместна работа с точно дефинирана и достижима цел, в която водеща роля има финтех компанията.

При дигитализацията на банковите услуги дейността на банковите служители не се ограничава до един банков клон, град, държава или територия. На практика чрез средствата на Интернет банковия служител е териториално независим - той може да е на едно място, а да извършва своята дейност на друго място (например, да контактува с клиент в друг град). Физическото териториално разделяне на служителите дава възможност оптимално да се подбере екип за работа, тъй като във всеки регион има потенциални служители с различна степен на квалификация и различно ниво на заплащане. Поради особеностите на тази форма на организация, използваните информационни технологии са от висока степен на важност за успеха на дигиталните услуги. Те оказват влияние както върху скоростта, с която се извършва процесът на обмен на информация, така и върху

способността на банката и финтех компанията да се възползват максимално от синергетичния ефект.

Основен компонент при дигитализация на банковите услуги представлява уеб базираната информационна система, чрез която работят и служителите и клиентите (Sajić et al., 2017). Връзката между тях се осъществява чрез световната мрежа Интернет, поради ниската цена за осъществяване на комуникацията и възможността за териториална независимост както на служителя, така и на клиента на банковата услуга.

От гледна точка на клиента на дигитализираните банкови услуги, те могат да се използват по два основни начина: стационарно и мобилно.

При стационарното използване на дигитализирани банкови услуги се използва декстоп или преносим компютър. Физическата връзка към Интернет може да е организирана по различни начини - най-често чрез LAN-карта или безжична Wi-Fi връзка, което дава възможност услугата да се ползва при относително ниски разходи.

При мобилното използване на дигитализирани банкови услуги липсата на стационарност се явява при някои ситуации едно от най-големите предимства. Чрез смартфон или таблет, физическата връзка към Интернет се осъществява най-често с безжична Wi-Fi връзка или 3/4/5G мобилен интернет, предлаган от мобилния оператор.

В зависимост от правата на достъп потребителят може да има право на активен достъп - да извършва разпоредителни операции, или на пасивен достъп - само да преглежда състоянието на активите си.

Дейността на финтех компаниите зависи изцяло от информационните технологии. В случай на стартъп се преминава през няколко етапа на придобиване и използване на информационни технологии за нуждите на своята дейност. През първия етап основно се закупува хардуер (изчислителна, комуникационна и др. техника) и софтуер (програмно осигуряване - системен, приложен, информационни

системи за управление и др.). През втория етап протича процес на поддържане на използвания хардуер и софтуер. На трети етап се появява нуждата от обновяване на използваните информационни технологии.

При сравнително малки финтех компании интуитивното вземане на решения от мениджърите по отношение на информационните технологии в повечето случаи е достатъчно за нормалното функциониране.

При сравнително големи финтех компании, със сложна организационна структура е твърде вероятно да започват да се появяват т.нар. "тесни места" - в тях изпълнението на някои процеси се задържа поради недостатъчен капацитет за обработка на проблема (Kitsios&Kamariotou, 2019). Това оказва съществено влияние върху процесите, които технологично са след "тесните места", тъй като се загубва синхронността при съвместната работата на отделните структурни звена. Появява се ситуация, при която служителите и мениджърите на ниски и средни нива на управление са неудовлетворени от работата си, въпреки наличието на съвременни информационни технологии. Възникват редица въпроси - например как ще се развива информационната система и има ли смисъл да се инвестира в новопоявяващи се технологии. Поради подобна неопределеност, мениджърите и служителите във финтех компанията може да не могат да осмислят достатъчно добре целите, задачите, принципите на функциониране и поради това да не могат да оптимизират всички аспекти от дейността си (Broby, 2019).

Появата на подобен проблем - разминаване на извършваната дейност с бизнес стратегията на финтех компанията, може да доведе до увеличаване на рисковете по всички направления: от една страна, при оперативната дейност - неспособност за справяне със заплахи към информационната сигурност - умишлена повреда на техника, хакерски атаки, вируси, кражба на данни и пр. От друга страна тези рискове може

да са от тактическо естество - при анализа, проектирането, документирането, закупуването и внедряването на информационни технологии. Възможно е да не се избере оптимално решение по отношение на информационните технологии, да не се използват напълно потенциалните възможности на внедрените информационни технологии, да има несъответствия между изградената инфраструктура (Амор, 2000, с.140) и насоките на развитие на информационните технологии, да има организационни и технически грешки при инсталирането на различни информационни системи, да не може да се поддържа и развива информационната система на приемливо ниво, да няма предвидени мероприятия при извънредни ситуации, да не може инфраструктурата да издържа нормални или пикови натоварвания, да не се прилагат неоптимални процедури по техническа поддръжка и др.

Намаляването на риска от подобни заплахи, които възпрепятстват нормалната дейност, а от там и постигането на стратегическите бизнес цели (Rehman&Anwar, 2019), е възможно да се постигне чрез създаването на Стратегия за развитие на информационните технологии във формален явен писмен вид, която да дефинира дългосрочните цели и "посоката на движение" в областта на информационните технологии. Стратегията за развитие на информационните технологии би следвало да е част от бизнес стратегията и да усилва бизнес предимствата на финтех компанията спрямо конкурентите. Нейното прилагане би трябвало да способства за успешното съществуване и развитие на компанията, като конкурентноспособен субект в една силно динамична пазарна среда. В този смисъл, можем да обобщим, че докато бизнес стратегията определя какво е необходимо да прави компанията, то Стратегията за развитие на информационните технологии отговаря за това как от гледна точка на информационните технологии би следвало да се реализира бизнес стратегията.

При разработването на Стратегия за развитие на

информационните технологии е необходимо да се отчита степента на важност на информационните технологии за отделните бизнес дейности (Ilmudeen et al., 2019) и да се осигури висока степен на непрекъсната работа на информационната инфраструктура и информационните услуги, тъй като при финтех компаниите те са от високо значение за основните бизнес процеси - електронна търговия, онлайн банкиране, интернет услуги и др. Като част от общата стратегия на компанията, Стратегията за развитие на информационните технологии е необходимо да указва по какъв начин на базата на актуални към момента информационни технологии могат да се подобрят бизнес процесите с цел увеличаване на печалбата - намаляване на разходите или увеличаване на приходите, и придобиване на конкурентни предимства. В идеалния случай задачата на Стратегията за развитие на информационните технологии е да намалява оперативните разходи за информационната инфраструктура, като за сметка на тях увеличава тези за внедряването на нови технологии за подкрепа на бизнес дейностите.

При големи финтех компании, според нас, е подходящо с цел по-голяма ефективност на разработената Стратегия за развитие на информационните технологии, предварително да се разработят отделни стратегии по основните дейности на компанията, след което Стратегията за развитие на информационните технологии е необходимо да ги осигурява и оптимизира. По този начин Стратегията за развитие на информационните технологии се съобразява с дългосрочните и средносрочните потребности на бизнеса, което позволява да се повиши ефективността на служителите, а от там и на компанията като цяло, по отношение на вземане на решения, работа с доставчици и клиенти, съвместна работа вътре в компанията и др.

Въз основа на направеното по-горе изложение, издигаме тезата, че при създаване на продукт или услуга от финтех компаниите не е подходящо създаването на интегрирани, с висока степен на сложност,

"монолитни" информационни системи, в които се прави опит да се обхванат всички бизнес процеси, тъй като времето за изграждане на подобни системи е много голямо. В периода от време между проектирането на продукта или услугата и нейното внедряване могат да се сменят не само хардуерните, софтуерните и комуникационните средства, които се използват при изграждането на информационната система, но и бизнес процесите - начините на управление, характерът на дейността и т.н., във финтех компанията. Това би наложило постоянен реинженеринг на продукта или услугата, което би увеличило значително разходите за разработка и поддръжка. Поради тези причини, модулното изграждане на информационната система, която стои зад продукта или услугата на финтех компанията е по-ефективно. Това е обосновано и с оглед на възприемането, че това са едни организации с относително по-кратък "живот". В идеалния случай финтех компанията би следвало да използва в комбинация няколко уеб базирани продукта от тип свободен софтуер.

Използването на готови стандартни решения от тип свободен софтуер за реализацията на продукти или услуги на финтех компании (например софтуер, разработван по проекта Fintech Open Source Foundation - https://www.finos.org/) може значително да опрости първоначалните етапи при изграждането на финтех продукта или услугата, а и в последствие да носи по-големи ползи и предимства за бизнеса. Някои от ползите при използването на готови стандартни решения от тип свободен софтуер са следните: намалява се зависимостта от доставчици на софтуер и хардуер; организацията на работа при нужда може лесно да се променя чрез смяна и доработка на различни софтуерни модули, тъй като изходния код е наличен и лицензa позволява това; на външни партньори може относително лесно да се предостави достъп до информация за състоянието на проектите, по които се работи; лесен обмен на данни между различни приложения,

благодарение на използването на общоприети стандарти.

Допълнителните предимства при използването на готови стандартни решения от тип свободен софтуер са следните: технологиите, на които се базира голяма част от свободния софтуер, са открити и се базират на документирани, стандартизирани, общо- и леснодостъпни протоколи и файлови формати; минимални разходи при изграждане и обновяване, тъй като по-голямата част от свободния софтуер е безплатен и освен с изходния си код, се разпространява и с правото да се използва, копира, модифицира и разпространява без да се иска разрешение от носителя на авторското право; висока сигурност и отказоустойчивост, тъй като се използва софтуер, програмни библиотеки и технологии, които са се доказали в практиката.

ГЛАВА ВТОРА. ИЗСЛЕДВАНЕ НА УЕБ ТЕХНОЛОГИИТЕ ЗА СИГУРНОСТ, ИЗПОЛЗВАНИ ОТ БАНКИТЕ

2.1. Изследване на банки от балтийските държави

2.1.1. Естония

Естония е една от водещите държави не само в ЕС, но и в света по прилагането на информационни технологии в държавното управление, финансовата сфера, бизнеса и обществения живот като цяло (Tammpuu&Masso, 2018; Kattel&Mergel, 2018). В Естония се предлагат голям набор от електронни услуги, които гражданите и бизнеса използват в ежедневието си. Налични са над 600 електронни услуги за гражданите и 2400 услуги за бизнеса по т.нар. проект "e-Estonia" (информацията е от уеб сайта <https://e-estonia.com/>). Характерно за личните карти, използвани в Естония е, че те могат да се ползват за нуждите на осигуряване на електронен подпис и по този начин онлайн могат да извършват редица услуги като подписване на договори, гласуване, Интернет банкиране, електронни рецепти, подаване на данъчни декларации и др.

В Естония е стартирала първата в света програма за електронно гражданство, която дава възможност на всеки гражданин на света, след издаването на цифров документ за идентификация, за регистрация и управление на фирма през Интернет. Според прогнози се очаква броят на "електронните граждани" до 2025 г. да достигне 10 милиона при собствено население от 1,3 милиона души. Използването на Интернет е значително - над 90% от естонците го използват (Roonemaa, 2017).

Страната е лидер в прилагането на нови информационни технологии в държавното управление, а от там и в бизнеса. Редица други държави също правят опити в подобна насока, но засега не толкова успешно (Skvarciany&Jurevičienė, 2017). Тъй като страната не е в челните позиции на богатите държави по БВП на глава от населението,

то би трябвало да се очаква, че успехите не се дължат толкова на абсолютния размер на финансовите средства, инвестирани в начинанието, а по скоро на високата им ефективност, което води до стабилни резултати. Доброто обмисляне преди отделянето на финансови средства е важен фактор за успеха от приложението на иновативни информационни технологии. В таблица 2.1.1 са представени данните за някои базови показатели за Естония.

Таблица 2.1.1. Демографски, макроикономически и други показатели за Естония

Показател	**Стойност**
Парична единица	евро
Население (2018) [милиони]	1,3
Площ [км^2]	45 339
БВП (номинален) на глава от населението (2018) [€]	19 500
Индекс за доходно неравенство Gini (2015)	32,7
Индекс на човешкото развитие HDI (2018)	0,871 (много висок)

(Източници: Eurostat, 2019; UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Основна роля в тези процеси играе банковата система и би било полезно събирането на данни за това какви уеб технологии се използват в публичните сайтове на банките в Естония. След като данните се обобщят, систематизират и анализират, информацията може да послужи като ръководство за добри практики при изграждане на стратегия и тактика за публично достъпни уеб сайтове и в други сектори.

Представените данните са събрани през месец април 2018 г.

Обхватът на изследването е ограничен до публичния сайт на конкретната банка, при което са изследвани сайтовете на 8 естонски банки (Petrov, 2019), лицензирани от Агенцията по финансов надзор на Естония (Finantsinspektsioon). Сайтовете на банки, които са лицензирани в други държави, но опериращи на база уведомителен режим и не се контролират от Естонската държава, не са разглеждани. Списъкът с лицензирани банки е получен от сайта на Агенцията по финансов надзор на Естония (Finantsinspektsioon, 2018), както и списъкът с адресите на банковите уеб сайтове (Таблица 2.1.2).

Таблица 2.1.2. Списък с естонските банки и техните адреси, лицензирани от Агенцията по финансов надзор на Естония към април 2018 г..

№	Официално име на банката според регистъра	URL адрес
1	AS Inbank	http://www.inbank.ee
2	AS LHV Pank	http://www.lhv.ee
3	AS Luminor Bank	http://www.luminor.ee/
4	AS SEB Pank	http://www.seb.ee
5	Bigbank AS	http://www.bigbank.ee/
6	Coop Pank aktsiaselts	http://www.cooppank.ee
7	Swedbank AS	http://www.swedbank.ee
8	TALLINNA ÄRIPANGA AS	http://www.tbb.ee

При събирането на данните е използван браузър Mozilla Firefox 60.0 - последна версия към момента на извършване на проучването.

За подобряване на уеб сигурността широко е разпространено да се използват допълнителни полета в заглавната част на отговора, който връща уеб сървърът. Тези полета указват на браузърите, които ги поддържат (почти всички съвременни браузъри поддържат тези полета)

да избягват потенциално опасни ситуации. В таблица 2.1.3 са представени наличието на съответните полета и техните стойности в различните банкови сайтове.

Таблица 2.1.3. Съдържание на различни полета, свързани със сигурността в заглавната част на отговора към април 2018 г.

Банка №	**Полета, свързани със сигурността в HTTP отговора**			
	Strict-Transport-Security	**X-Frame-Options**	**X-XSS-Protection**	**X-Content-Type-Options**
1	max-age=16000000; includeSubDomains; preload;	-	-	-
2	-	-	-	-
3	max-age=1000	SameOrigin	1; mode=block	nosniff
4	-	-	-	-
5	max-age=31536000; includeSubDomains; preload		1; mode=block	-
6	-	SAMEORIGIN	-	nosniff
7	max-age=31536000; includeSubDomains	SAMEORIGIN	1; mode=block	nosniff
8	-	SAMEORIGIN	-	-

Данните в таблица 2.1.3 показват, че по-голяма част от уеб сайтовете използват полета в HTTP отговора, които са свързани със сигурността - 25% не използват такива полета (банки №2 и 4), 25% използват всички възможни полета (банки №3 и 7), а 50% използват

само една част от полетата. Половината - 50% от сървърите поддържат полета Strict-Transport-Security и X-Frame-Options, а 38% поддържат полетата X-XSS-Protection и X-Content-Type-Options.

Наличието на полето "Strict-Transport-Security", ако се поддържа от браузъра, ще предотврати комуникацията със зададения домейн да се извършва по HTTP вместо по HTTPS (Hodges et al., 2012). Полето "X-Frame-Options" в заглавната част на отговора осигурява защита срещу така наречената "атака на Clickjack". Параметърът "SAMEORIGIN" инструктира браузъра да показва уеб страницата в HTML тагове <frame> ... <iframe> само ако домейнът на двете уеб страници е еднакъв (Ross et al., 2013).

Полето "X-XSS-Protection" осигурява защита срещу т.нар. "Cross-Site Scripting Attack". Параметърът "1; mode = block" указва на браузъра да блокира показването на уеб страницата, ако бъде установена такъв вид атака. Полето "X-Content-Type-Options" указва на браузърът да използва само изпратения в полето "Content-Type" параметър за определяне на типа на изпратените данни.

Данни за максималната поддържана версия на HTTP и съдържанието на полетата, идентифициращи сървъра, както и настройките за поддържане на мрежовата връзка, отворена за по-продължителен период от време, са показани в таблица 2.1.4.

Таблица 2.1.4. Версия на HTTP и съдържанието на полетата, идентифициращи сървъра и указващи поддържането на отворена мрежова връзка към април 2018 г.

Банка №	**Версия на HTTP**	**Сървър**	**Keep-Alive Timeout**	**Maximum served files**
1	1.1	-	-	-
2	1.1	Apache	5	100
3	1.1	-	5	100

4	1.1	Apache	-	-
5	1.1	Apache/2.4.12 (Red Hat)	-	-
6	2	Apache/CloudFront	-	-
7	1.1	-	-	-
8	1.1	-	20	-

Нашите проучвания показват, че половината от администраторите са задали настройки на конфигурационните файлове на уеб сървъра да не подават информация кой е и каква е неговата версия (Петров, 2018). За останалите 50% от уеб сървърите може да се предполага, че са Apache, но трябва да се има в предвид, че тази информация може умишлено да се подмени от системните администратори с цел заблуда. Сървърите на голяма част от банките, които работят по HTTP 1.1 не поддържат мрежовата връзка отворена, за да обслужат по нея и други заявки. Този подход обикновено има за цел да предотврати ситуациите от тип Deny of Service, при които атакуващият отваря едновременно голям брой мрежови връзки, които се поддържат отворени дълъг период от време, което от своя страна предизвиква ситуация на отказ от обслужване на обикновените клиенти.

Интересно е, че само една естонска банка (банка №6) използва най-новия протокол HTTP/2, които може да реши редица от проблемите, свързани с латенцията при зареждане на уеб страници. Както вече посочихме в Първа глава, той се различава съществено от предходния протокол версия 1.1 и той притежава множество подобрения спрямо 1.1.

В таблица 2.1.5 обобщено са представени данни по основните показатели за използването на HTTPS от естонските банки. Тенденцията през последните години е все повече HTTPS да се използва като подразбиращ се протокол при задаване на хипервръзки в уеб страници и да се използва по подразбиране от уеб приложенията. Дори клиентът да

направи опит за връзка по некриптирана връзка, той автоматично се пренасочва да използва криптирана. Естонските банки следват тази глобална тенденция, като сайтът само на една от изследваните банки (банка №8) не пренасочва автоматично към криптирана връзка.

Таблица 2.1.5. Използване на HTTPS от естонските банки към април 2018 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	EV	GeoTrust EV SSL CA - G4	14
2	да	EV	Thawte Extended Validation SHA256 SSL CA	26
3	да	DV	Let's Encrypt Authority X3	3
4	да	EV	GlobalSign EV CA - SHA256 - G3	13
5	да	EV	Thawte EV RSA CA 2018	6
6	да	DV	Amazon Server CA 1B	13
7	да	EV	Symantec Class 3 EV SSL CA - G3	12
8	не	DV	DigiCert SHA2 Secure Server CA	36

Нашето проучване установи, че множеството от издатели на сертификати е много добре диверсифицирано, но могат да се определят изявени пазарни лидери в този сегмент на естонския ИТ пазар. Водещо място имат сертификатите на американската компания Symantec - 1 бр.,

заедно с дъщерните й компании Thawte - 2 бр. и GeoTrust – 1 бр. Като резултат, общият пазарен дял на групата на Symantec (банки №1, 2, 5, 7) е 50%. За другите издатели - GlobalSign и DigiCert няма данни за пряка свързаност с други компании (има косвена свързаност - компанията Thoma Bravo е собственик на DigiCert и на част от бизнеса на Symantec за издаване на сертификати). Интерес представлява използването на облачните услуги на Amazon от една от банките - обикновено банките разчитат на собствена ИТ инфраструктура. Друг интересен момент е използването на безплатни сертификати от Let's Encrypt от банка №3 - обикновено банките имат достатъчно финансови ресурси, за да закупят сертификат от реномиран издател.

Нашето проучване също така установи, че 63% от банките използват EV, а 37% - DV. Допълнително нашето проучване установи, че разнообразието от периоди на валидност на сертификати е голямо - варира от 3 месеца до 3 г., като средният период на валидност на SSL/TSL сертификатите, използвани от банките е 15,4 месеца. Може да се приеме, че усреднено период от около 1 година до 1,5 години за валидност на сертификата е добър избор.

В изследването бяха проучени уеб сайтовете на 8 естонски банки, подлежащи на контрол от местния регулаторен орган. Установиха се следните по-важни факти:

1. 25% от уеб сървърите на банките не използват полета в заглавната част на отговорите, свързани със сигурността; 25% използват всички възможни полета; 50% използват само една част от полетата.

2. Уеб сървърът само на една естонска банка използва най-новия протокол HTTP/2. Уеб сървърът само на една банка не пренасочва автоматично към криптирана (HTTPS) връзка.

3. При SSL/TLS сертификатите общият пазарен дял на групата на Symantec е 50%. Една банка използва безплатен сертификат от Let's Encrypt.

4. 63% от банките използват разширено валидиране, а 37% само валидиране на домейн при издаване на SSL/TLS сертификатите. Периодите на валидност на сертификатите варират в широки граници - от 3 месеца до 3 години.

След период от 16 месеца, през август 2019 г. изследването беше повторено по част от изследваните показатели и в него беше включена още една банка, започнала да оперира на местния финансов пазар под контрола на местния държавен финансов и банков орган - Estonia Finantsinspektsioon. Списъците на банките, оторизирани да оперират в Естония, отново са взети от там - таблица 2.1.6 и новите данни са представени в таблица 2.1.7.

Периодът от време от 16 месеца беше избран, тъй като беше установено, че средният период на валидност на SSL/TSL сертификатите, използвани от банките е 15,4 месеца, при което направихме предположението, че по-голямата част от банките в този период ще са сменили/подновили своите сертификати.

Таблица 2.1.6. Използване на протокол HTTPS от естонски банки към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	AS Inbank	www.inbank.ee	да
2	AS LHV Pank	www.lhv.ee	да
3	AS Luminor Bank	www.luminor.ee	да
4	AS SEB Pank	www.seb.ee	да
5	AS TBB pank	www.tbb.ee	да
6	Bigbank AS	www.bigbank.ee	да
7	Coop Pank aktsiaselts	www.cooppank.ee	да
8	Holm Bank AS	www.holmbank.ee	да
9	Swedbank AS	www.swedbank.ee	да

При сравнение на данните в таблица 2.1.2 и таблица 2.1.6 се вижда, че една банка е сменила името си, без да променя името на домейна (банка №8 от таблица 2.1.2 и №5 от таблица 2.1.6) и има една нова банка (банка №8 в таблица 2.1.6).

Таблица 2.1.7. Основни параметри при използване на HTTPS от естонски банки към август 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	DV	Sectigo RSA Domain Validation Secure Server CA	26
2	да	EV	DigiCert SHA2 Extended Validation Server CA	27
3	да	DV	Let's Encrypt Authority X3	3
4	да	EV	GlobalSign Extended Validation CA - SHA256 - G3	13
5	НЕ	DV	DigiCert SHA2 Secure Server CA	25
6	да	EV	DigiCert SHA2 Extended Validation Server CA	26
7	да	DV	Amazon	13
8	да	EV	Sectigo RSA Extended Validation Secure	12

			Server CA	
9	да	EV	DigiCert SHA2 Extended Validation Server CA	12

При анализа на данните, събрани през април 2018 г. и през август 2019 г. се установи следното:

1. Средният период на валидност на SSL/TSL сертификатите, използвани от банките, се е повишил от 15,4 месеца на 17,4 месеца.

2. Броят на EV сертификатите се е запазил - 5 бр., а на DV сертификатите се е увеличили от 3 на 4. Новодобавената банка е със сертификат EV, а друга една банка преминала от 14 месечен EV на 26 месечен DV сертификат, което представлява регрес по отношение на спазването на добрите практики.

3. Значително се засилва консолидацията на пазара на SSL/TSL сертификати - увеличава се пазарния дял на DigiCert. Това се дължи на обстоятелството, че през 2017 г. компанията Thoma Bravo, собственик на DigiCert купува бизнеса на Symantec за SSL/TSL сертификати (Symantec Identity Services), а Symantec, от своя страна е придобила Thawte и GeoTrust от Verisign през 2010 г. В резултат, сертификатите, издадени от DigiCert нарастват от 1 към април 2018 г. на 4 броя към август 2019 г.

4. Появява се нов играч на този пазарен сегмент - Sectigo (бивше Comodo преди 2018 г.), чиито сертификати се ползват от 2 банки към август 2019 г.

2.1.2. Латвия

В проучването са обхванати началните страници на 14 латвийски банки през август 2019 г. Списъкът с банките, оторизирани да оперират в Латвия са взети от уеб сайта на Комисията за финансов и капиталов пазар на Латвия (Latvia Financial And Capital Market Commission, 2019).

В проучването са изключени уеб сайтове на чуждестранни банкови клонове и представителства на чуждестранни банки, опериращи на местните финансови пазари. Проучени са само местни банки, които работят под регулацията на контролния орган на държавата, а уеб сайтовете на банки, които работят на трансгранична основа в ЕС, не са включени. В таблица 2.1.8 са представени данните за някои базови показатели за Латвия, в таблица 2.1.9 - списъка с изследваните сайтове на латвийски банки, и в таблица 2.1.10 - основните параметри при използване на HTTPS от латвийски банки.

Таблица 2.1.8. Демографски, макроикономически и други показатели за Латвия

Показател	**Стойност**
Парична единица	евро
Население (2018) [милиони]	1,9
Площ [км^2]	64 589
БВП (номинален) на глава от населението (2018) [€]	15 300
Индекс за доходно неравенство Gini (2015)	34,2
Индекс на човешкото развитие HDI (2018)	0,847 (много висок)

(Източници: Eurostat, 2019; UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Основният метод, използван в проучването, включва анализ на отговорите, дадени от уеб сървърите. Актуализиран браузър Google Chrome версия 76.0.3809.100 (Official Build) (64-битов), работещ под типичен настолен компютър с Windows 10 Professional Edition x64, беше използван като уеб клиент с активиран модул "Developer Tools". Процесът на проверка е извършен ръчно чрез експертна оценка. Други подходи за извършване на същите изследвания биха могли да включват

използването на инструменти на командния ред като "curl", но използването на уеб браузър води до по-точни резултати.

Таблица 2.1.9. Използване на протокол HTTPS от латвийски банки към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	AS Baltic International Bank	www.bib.lv, www.bib.eu	да
2	AS Citadele banka	www.citadele.lv	да
3	AS LPB Bank	www.lpb.lv	НЕ
4	AS Reģionālā investīciju banka	www.ribbank.com	НЕ
5	AS Rietumu Banka	www.rietumu.lv	да
6	AS Meridian Trade Bank	www.mtbank.eu	да
7	AS PrivatBank	www.privatbank.lv	да
8	AS BlueOrange Bank	www.blueorangebank.com	да
9	AS Expobank	www.expobank.eu	да
10	AS PNB Banka	www.pnbbanka.eu	да
11	AS SEB banka	www.seb.lv	да
12	Rigensis Bank AS	www.rigensisbank.com	да, но пренасочва към HTTP
13	Signet Bank AS	www.signetbank.com	да
14	Swedbank AS	www.swedbank.lv	да

Таблица 2.1.10. Основни параметри при използване на HTTPS от латвийски банки към август 2019 г.

Банка №	Автоматично пренасочване към HTTPS	Тип	Издател на SSL/TLS сертификат	Валидност в месеци
1	да	EV	DigiCert SHA2 Extended	25

			Validation Server CA	
2	да	EV	Thawte EV RSA CA 2018	24
5	да	DV	DigiCert SHA2 Secure Server CA	26
6	да	EV	Thawte EV RSA CA 2018	25
7	да	DV	Go Daddy Secure Certificate Authority - G2	13
8	да	DV	DigiCert SHA2 Secure Server CA	24
9	да	DV	Let's Encrypt Authority X3	3
10	да	DV	Thawte RSA CA 2018	12
11	да	DV	GlobalSign Organization Validation CA - SHA256 - G2	25
12	НЕ	DV	DigiCert SHA2 Secure Server CA	15
13	да	DV	Go Daddy Secure Certificate Authority - G2	24
14	да	EV	DigiCert SHA2 Extended Validation Server CA	12

Уебсайтовете на три банки в Латвия не използват HTTPS (№3, №4 и №12) - две не използват HTTPS изобщо (№3 и №4), а в един случай (№12) HTTPS заявките се пренасочват за използване на HTTP. Броят на уеб сайтовете на банките, които не използват HTTPS, не е голям, но тази ситуация е доста странна, тъй като цените за обикновен DV сертификат започват от около 30 евро годишно и също така има безплатна алтернатива. Авторитетни организации и компании, като Electronic Frontier Foundation, Mozilla, Akamai, Cisco, IdenTrust и други, съвместно създадоха сертифициращ орган Let's Encrypt, с основната цел да се издават безплатни сертификати. Понастоящем тези сертификати са

с период на валидност само 3 месеца.

Относно случая на пренасочване на заявките от HTTPS към използване на HTTP (латвийска банка №12) ще бъде по-добре или да се поддържа HTTPS според добрите практики, или изобщо да не се използва HTTPS, защото тези проблеми биха могли да отслабят доверието на клиентите в банката към способността да поддържа своите системи.

2.1.3. Литва

В проучването бяха обхванати 4 литовски банки през август 2019 г. Основният метод, използван в проучването, е описан в предходната точка. Методиката на изследването се основава частично на методологията, използвана в предишни проучвания (Petrov&Hundal, 2018; Petrov et al., 2018a; Petrov et al., 2019a) на уеб технологиите, използвани в банките. Списъкът с банките, оторизирани да оперират в Литва е взет уеб сайта на централната банка на Литва (Bank of Lithuania, 2019). Изключени са уеб сайтове на чуждестранни банкови клонове и представителства на чуждестранни банки. В таблица 2.1.11 са представени данните за някои базови показатели за Литва, а в таблица 2.1.12 - списъка с изследваните сайтове на литовски банки.

Таблица 2.1.11. Демографски, макроикономически и други показатели за Литва

Показател	**Стойност**
Парична единица	евро
Население (2018) [милиони]	2,8
Площ [км^2]	65 300
БВП (номинален) на глава от населението (2018) [€]	16 100
Индекс за доходно	37,4

неравенство Gini (2015)	
Индекс на човешкото развитие HDI (2018)	0,858 (много висок)

(Източници: Eurostat, 2019; UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Таблица 2.1.12. Използване на протокол HTTPS от литовски банки към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	Akcinė bendrovė Šiaulių bankas	www.sb.lt	да
2	AB SEB bankas	www.seb.lt	да
3	Swedbank AB	www.swedbank.lt	да
4	UAB Medicinos bankas	www.medbank.lt	да

Обобщените резултати от изследваните уеб страници са представени в следващата таблица (таблица 2.1.13) въз основа на следните ключови показатели: наличие на автоматично пренасочване към HTTPS, вид на сертификата, име на сертифициращия орган и срок на валидност на SSL/TLS сертификата.

Таблица 2.1.13. Основни параметри при използване на HTTPS от литовски банки към август 2019 г.

Банка №	Автоматично пренасочване към HTTPS	Тип	Издател на SSL/TLS сертификат	Валидност в месеци
1	да	DV	Thawte TLS RSA CA G1	19
2	да	DV	GlobalSign Organization Validation CA - SHA256 - G2	25

3	да	EV	DigiCert SHA2 Extended Validation Server CA	12
4	да	DV	COMODO RSA Domain Validation Secure Server CA	36

2.1.4. Сравнителен анализ между балтийските държави

Данните, представени в таблица 2.1.7, таблица 2.1.10 и таблица 2.1.13 дават съвкупна представа за състоянието в трите балтийски държави Естония, Латвия и Литва по отношение използването на HTTPS към август 2019 г. Данните за тип на сертификата са обобщени в таблица 2.1.14. Става ясно, че мнозинството - 15 балтийски банки използват обикновени DV сертификати, а само 10 балтийски банки използват по-сложните за издаване на EV, които са и по-авторитетни. Само в Естония мнозинството (56%) от банките използват EV сертификати, докато в Латвия и Литва ситуацията е обратна - мнозинството (57% в Латвия и 75% в Литва) използват DV.

Таблица 2.1.14. Типове сертификати, използвани в публичните сайтове на банки от балтийските държави към август 2019 г.

Тип	**Естония**		**Латвия**		**Литва**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
без сертификат	-	-	2	14	-	-
DV	4	44	8	57	3	75
EV	5	56	4	29	1	25
ОБЩО	*9*	*100*	*14*	*100*	*4*	*100*

В два случая - една банка от Естония и една банка от Латвия, добрите практики не се спазват и HTTP заявките не се пренасочват

автоматично, за да се използва защитена HTTPS връзка.

Има голямо разнообразие от предпочитания за сертифициращ орган, но най-популярните избори в балтийските държави са:

• DigiCert - 10 банки;

• Thawte - 4 банки;

• GlobalSign - 3 банки;

• Sectigo / Comodo (преди 2018 г. Sectigo е била Comodo) - 3 банки;

• Go Daddy - 2 банки;

• Let's Encrypt (безплатни 3-месечни сертификати) - 2 банки;

• Amazon - 1 банка.

Данните за сертифициращия орган, които са представени в таблица 2.1.7, таблица 2.1.10 и таблица 2.1.13, са обобщени за удобство в таблица 2.1.15.

Таблица 2.1.15. Издатели на сертификати, използвани в публичните сайтове на банки от балтийските държави към август 2019 г.

Издател	**Естония**		**Латвия**		**Литва**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
Amazon	1	11	-	-	-	-
DigiCert	4	45	5	42	1	25
GlobalSign	1	11	1	8	1	25
Go Daddy	-	-	2	17	-	-
Let's Encrypt	1	11	1	8	-	-
Sectigo/Comodo	2	22	-	-	1	25
Thawte	-	-	3	25	1	25
TOTAL	*9*	*100*	*12*	*100*	*4*	*100*

Сравнението между трите балтийски държави води до следните

заключения. Първо, банковият сектор в Литва, която е най-голямата балтийска държава, е по-консолидиран от този в Естония и Латвия. Второ, що се отнася до използването на SSL/TLS сертификати, най-популярният доставчик на SSL/TLS сертификати е DigiCert с дял от 40% уеб сайтове. Интересно е, че една естонска и една латвийска банка използват безплатни сертификати от Let's Encrypt Authority. Една банка в Естония, две банки в Латвия не се пренасочват автоматично от несигурен HTTP за осигуряване на HTTPS връзка. Три банки в Латвия изобщо не използват SSL/TLS или не пренасочват към сигурна HTTPS връзка при първоначално използвана несигурна HTTP връзка. Последното считаме за недобра практика.

Средната валидност на сертификатите е 1 година и 7 месеца, с медиана - 1 година и 11 месеца.

Събраните данни са свързани с конкретен период - август 2019 г. Резултатите от проучването биха могли да окажат важно практическо въздействие за банковите мениджъри и ИТ специалистите при оценка на възможностите, кои технологии да се прилагат, за да се сведе до минимум риска за финансовата институция. Освен това резултатите разкриват някои добри и лоши практики, използвани в банките на балтийските държави. Изследването, проведено върху използването на протокола HTTPS на публичните уебсайтове на банките, обхвана сайтовете на всички 9 естонски, 14 латвийски и 4 литовски банки, лицензирани да оперират на съответната територия на страната от местните национални банки или друга държавна институция.

2.2. Изследване на банки от централно европейски държави

2.2.1. Чехия

Чешката република (Чехия) се намира в Централна Европа и в миналото, заедно със Словакия са били една обща държава. Чехия използва своя собствена парична единица. Това дава възможност

банковата система да е формално независима от по-високо равнище на надзор от страна на Европейската централна банка. В таблица 2.2.1 са представени данните за някои базови демографски, макроикономически и други показатели за Чехия.

Таблица 2.2.1. Демографски, макроикономически и други показатели за Чехия

Показател	**Стойност**
Парична единица	CZK
Население (2018) [милиони]	10,6
Площ [км2]	78 866
БВП (номинален) на глава от населението (2019 прогн.) [USD]	23 209
БНД на глава от населението (метод Atlas, 2018) [USD]	20 260
Индекс за доходно неравенство Gini (2018)	24,0
Индекс на човешкото развитие HDI (2017)	0,888 (много висок)
Рейтинг на S&P Global (дългосрочен, 2011)	AA-
Рейтниг на Moody's Investors Service (дългосрочен, 2018)	A1
Рейтниг на Fitch (дългосрочен, 2019)	AA-

Източници: Country Economy, 2019; Eurostat, 2019; Wikipedia, 2019; World Bank DRG, 2019; UN HDRO, 2019)

В проучването са обхванати уеб сайтовете на 22 чешки банки през август 2019 г. (Petrov et al., 2019b). Методологията на проучването

се основава отчасти на методологията, използвана в предходни проучвания на уеб технологии, използвани в банки (Petrov&Hundal, 2018; Petrov et al., 2018a; Petrov et al., 2019a). Списъкът на банките, на които е разрешено да оперират в Чехия, е взет от уеб сайта на Чешката национална банка (Czech National Bank, 2019) (таблица 2.2.2). Основните характеристики при използването на HTTPS в публични уеб сайтове на чешките банки към август 2019 г. са дадени в таблица 2.2.3.

Таблица 2.2.2. Използване на протокол HTTPS в публичните уеб сайтове на банки в Чешката република към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	Air Bank a.s.	airbank.cz	да
2	Banka CREDITAS a.s.	creditas.cz	да
3	Česká exportní banka, a.s.	ceb.cz	да
4	Česká spořitelna, a.s.	csas.cz	да
5	Českomoravská stavební spořitelna, a.s.	cmss.cz	да
6	Českomoravská záruční a rozvojová banka, a.s.	cmzrb.cz	да
7	Československá obchodní banka, a. s.	csob.cz	да
8	Equa bank a.s.	equabank.cz	да
9	Expobank CZ a.s.	expobank.cz	да
10	Fio banka, a.s.	fio.cz	да
11	Hypoteční banka, a.s.	hypotecnibanka.cz	да
12	Industrial and Commercial Bank of China Limited, Prague Branch, odštěpný závod	icbc-cz.com	НЕ
13	J & T BANKA, a.s.	jtbank.cz	да
14	Komerční banka, a.s.	kb.cz	да

15	Modrá pyramida stavební spořitelna, a.s.	modrapyramida.cz	да
16	MONETA Money Bank, a.s.	moneta.cz	да
17	PPF banka a.s.	ppfbanka.cz	да
18	Raiffeisen stavební spořitelna a.s.	rsts.cz	да
19	Raiffeisenbank a.s.	rb.cz	да
20	Sberbank CZ, a.s.	sberbankcz.cz	да
21	UniCredit Bank Czech Republic and Slovakia, a.s.	unicreditbank.cz	да
22	Wüstenrot hypoteční banka a.s./Wüstenrot - stavební spořitelna a.s.	wuestenrot.cz	да

Таблица 2.2.3. Основни характеристики при използването на HTTPS в публични уеб сайтове на чешките банки към август 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	EV	DigiCert SHA2 Extended Validation Server CA	26
2	да	EV	Thawte EV RSA CA 2018	26
3	да	EV	Thawte EV RSA CA 2018	26
4	да	EV	DigiCert SHA2 Extended Validation Server CA	14
5	да	EV	DigiCert SHA2 Extended Validation Server CA	24
6	да	EV	Thawte EV RSA CA 2018	12
7	да	EV	DigiCert SHA2 Extended Validation Server CA	24
8	да	EV	Thawte EV RSA CA 2018	22

9	да	DV	Thawte RSA CA 2018	14
10	да	EV	GeoTrust EV RSA CA 2018	27
11	да	EV	Thawte EV RSA CA 2018	26
12	НЕ	-	-	-
13	да	DV	Thawte TLS RSA CA G1	37
14	да	EV	DigiCert Global CA G2	12
15	да	DV	COMODO ECC Domain Validation Secure Server CA 2	6
16	да	EV	DigiCert Global CA G2	12
17	да	EV	Thawte EV RSA CA 2018	26
18	да	EV	Thawte EV RSA CA 2018	12
19	да	EV	DigiCert SHA2 Extended Validation Server CA	14
20	да	EV	Thawte EV RSA CA 2018	26
21	да	DV	Actalis Organization Validated Server CA G2	12
22	да	EV	Thawte EV RSA CA 2018	26

Един от изследваните чешки публични банки уеб сайтове не използва HTTPS (банка 12). Останалите посетени уеб сайтове на Чешката публична банка (95,5%) използват сертификати от признати сертифициращи органи (CAs), които гарантират, че публичният ключ, използван за шифроване на връзката съответства на името на домейна, съхранявани в DNS сървъри. Най-предпочитания тип SSL/TLS сертификат от чешките банки е EV - 77,3%, което е много добра практика.

2.2.2. Словакия

Словашката република (Словакия) се намира в Централна Европа

и в миналото заедно със Чехия са били една обща държава. За разлика от Чехия, която използва собствена парична единица, Словакия е приела еврото за своя парична единица. Съответно банковата система в страната е под надзор от страна на Европейската централна банка. В таблица 2.2.4 са представени данните за някои базови демографски, макроикономически и други показатели за Словакия.

Таблица 2.2.4. Демографски, макроикономически и други показатели за Словакия (Словашката република)

Показател	**Стойност**
Парична единица	евро
Население (2018) [милиони]	5,5
Площ [км2]	49 035
БВП (номинален) на глава от населението (2019 прогн.) [USD]	20 155
БНД на глава от населението (метод Atlas, 2018) [USD]	18 330
Индекс за доходно неравенство Gini (2018)	23,2
Индекс на човешкото развитие HDI (2017)	0,855 (много висок)
Рейтинг на S&P Global (дългосрочен, 2015)	A+
Рейтниг на Moody's Investors Service (дългосрочен, 2018)	A2
Рейтниг на Fitch (дългосрочен, 2019)	A+

Източници: Country Economy, 2019; Eurostat, 2019; Wikipedia, 2019; World Bank DRG, 2019; UN HDRO, 2019)

В проучването са обхванати уеб сайтовете на 15 словашки банки

през август 2019. Списъкът на банките, на които е разрешено да оперират в Словакия, е взет от уеб сайта на централната банка на Словения (Bank of Slovenia, 2019) (таблица 2.2.5). Основните характеристики при използването на HTTPS в публичните уеб сайтове на словашките банки към август 2019 г. са представени в таблица 2.2.6.

Таблица 2.2.5. Използване на протокол HTTPS в публичните уеб сайтове на банки в Словакия към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	Abanka d.d.	abanka.si	да
2	Addiko Bank d.d.	addiko.si	да
3	Banka Intesa Sanpaolo d.d.	intesasanpaolobank.si	да
4	Banka Sparkasse d.d.	sparkasse.si	да
5	Delavska hranilnica d.d.	delavska-hranilnica.si	да
6	Deželna banka Slovenije d.d.	dbs.si	да
7	Gorenjska banka d.d.	gbkr.si	да
8	Hranilnica LON d.d.	lon.si	да
9	Nova Kreditna banka Maribor d.d.	nkbm.si	да
10	Nova Ljubljanska banka d.d.	nlb.si	да
11	Primorska hranilnica Vipava d.d.	phv.si	да
12	Sberbank banka d.d.	sberbank.si	да
13	SID-Slovenska izvozna in razvojna banka, d.d.	sid.si	да
14	SKB banka d.d.	skb.si	да
15	UniCredit Banka Slovenija d.d.	unicredit.si	да

Таблица 2.2.6. Основни характеристики при използването на HTTPS в публичните уеб сайтове на словашките банки към август 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	yes	EV	DigiCert SHA2 Extended Validation Server CA	25
2	yes	EV	GeoTrust EV RSA CA 2018	15
3	yes	DV	DigiCert SHA2 Secure Server CA	25
4	yes	DV	COMODO RSA Domain Validation Secure Server CA	24
5	yes	EV	Thawte EV RSA CA 2018	25
6	yes	DV	Entrust Certification Authority - L1K	24
7	yes	DV	DigiCert SHA2 Secure Server CA	24
8	yes	DV	Thawte RSA CA 2018	18
9	yes	EV	GeoTrust EV RSA CA 2018	10
10	yes	EV	Entrust Certification Authority - L1M	24
11	yes	DV	Let's Encrypt Authority X3	3.
12	yes	DV	GeoTrust RSA CA 2018	14
13	NO	DV	DigiCert SHA2 Secure Server CA	24
14	yes	EV	Thawte EV RSA CA 2018	15
15	yes	DV	Actalis Organization	12

			Validated Server CA G2	

Всички изследвани уеб сайтове на словашките публични банки използват HTTPS и SSL/TLS сертификати. Най-предпочитания тип SSL/TLS сертификат е DV - 60%, което не е добра практика. Само 40% от банките използват сертификати на EV SSL/TLS. Една банка използва безплатен SSL/TLS сертификат.

2.2.3. Унгария

Унгария се намира в Централна Европа и подобно на Чехия използва своя собствена парична единица. Това дава възможност банковата система да е формално независима от по-високо равнище на надзор от страна на Европейската централна банка. В таблица 2.2.7 са представени данните за някои базови демографски, макроикономически и други показатели за Унгария.

Таблица 2.2.7. Демографски, макроикономически и други показатели за Унгария

Показател	**Стойност**
Парична единица	HUF
Население (2018) [милиони]	9,8
Площ [км2]	93 030
БВП (номинален) на глава от населението (2019 прогн.) [USD]	17 296
БНД на глава от населението (метод Atlas, 2018) [USD]	14 590
Индекс за доходно неравенство Gini (2018)	28,7
Индекс на човешкото развитие HDI	0,838 (много висок)

(2017)	
Рейтинг на S&P Global (дългосрочен, 2019)	BBB
Рейтниг на Moody's Investors Service (дългосрочен, 2018)	Baa3
Рейтниг на Fitch (дългосрочен, 2019)	BBB

Източници: Country Economy, 2019; Eurostat, 2019; Wikipedia, 2019; World Bank DRG, 2019; UN HDRO, 2019)

В проучването са обхванати уеб сайтовете на 22 унгарски банки през август 2019. Списъкът на банките, на които е разрешено да оперират в Унгария, е взет от уеб сайта на централната банка на Унгария (Magyar Nemzeti Bank, 2019) (таблица 2.2.8). Основните характеристики при използването на HTTPS в публичните уеб сайтове на унгарските банки към август 2019 г. са представени в таблица 2.2.9.

Таблица 2.2.8. Използване на протокол HTTPS в публичните уеб сайтове на банки в Унгария към август 2019 г.

№	Банка	Банков домейн	HTTPS
1	BUDAPEST Hitel- és Fejlesztési Bank Zrt.	budapestbank.hu	да
2	CIB Bank Zrt.	cib.hu	да
3	Commerzbank Zrt.	commerzbank.hu	да
4	DUNA TAKARÉK BANK Zrt.	dunatakarek.hu, dtbank.hu	да
5	ERSTE BANK HUNGARY Zrt.	erstebank.hu	да
6	GRÁNIT Bank Zrt.	granitbank.hu	да
7	KDB Bank Európa Zrt.	kdbbank.eu	да

8	Kereskedelmi és Hitelbank Zrt.	kh.hu	да
9	MagNet Magyar Közösségi Bank Zrt.	magnetbank.hu	да
10	Magyar Cetelem Bank Zrt.	cetelem.hu	да
11	Merkantil Váltó és Vagyonbefektető Bank Zrt.	merkantil.hu	да
12	MKB Bank Nyrt.	mkb.hu	да
13	MTB Magyar Takarékszövetkezeti Bank Zrt.	mtb.hu	НЕ
14	OTP Bank Nyrt.	otpbank.hu	да
15	Polgári Bank Zrt.	polgaribank.hu	да
16	Porsche Bank Hungaria Zártkörűen Működő Rt.	porschebank.hu	да
17	Raiffeisen Bank Zrt.	raiffeisen.hu	да
18	Sberbank Magyarország Zrt.	sberbank.hu	да
19	SOPRON BANK BURGENLAND Zrt.	sopronbank.hu	да
20	TAKARÉK Kereskedelmi Bank Zrt.	takarek.hu	да
21	Takarékbank Zrt.	takarekbank.hu	да
22	UniCredit Bank Hungary Zrt.	unicreditbank.hu	да

Таблица 2.2.9. Основни характеристики при използването на HTTPS в публични уеб сайтове на унгарските банки към август 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	EV	DigiCert SHA2 Extended	16

			Validation Server CA	
2	да	EV	DigiCert Global CA G2	24
3	да	EV	GlobalSign Extended Validation CA - SHA256 - G3	26
4	да	DV	NetLock Üzleti (Class B) Tanúsítványkiadó	24
5	да	DV	NetLock Üzleti (Class B) Tanúsítványkiadó	24
6	да	DV	NetLock Üzleti (Class B) Tanúsítványkiadó	12
7	да	DV	NetLock Közjegyzői (Class A) Tanúsítványkiadó	24
8	да	EV	DigiCert SHA2 Extended Validation Server CA	24
9	да	DV	NetLock Közjegyzői (Class A) Tanúsítványkiadó	24
10	да	DV	DigiCert SHA2 High Assurance Server CA	12
11	да	DV	NetLock Üzleti (Class B) Tanúsítványkiadó	24
12	да	EV	Sectigo RSA Extended Validation Secure Server CA	24
13	НЕ	-	-	-
14	да	EV	DigiCert SHA2 Extended Validation Server CA	12

15	да	DV	Let's Encrypt Authority X3	3
16	да	DV	Let's Encrypt Authority X3	3
17	да	EV	GeoTrust EV RSA CA 2018	23
18	да	DV	NetLock Üzleti (Class B) Tanúsítványkiadó	24
19	да	DV	RapidSSL RSA CA 2018	26
20	да	DV	Sectigo RSA Domain Validation Secure Server CA	12
21	да	DV	Let's Encrypt Authority X3	3
22	да	DV	Actalis Organization Validated Server CA G2	12

Един от изследваните уеб сайтове на унгарски банки не използва HTTPS (банка 13). Най-предпочитания тип SSL/TLS сертификат е DV - 63,6%, което не е добра практика. Само 31,8% от банките използват сертификати от тип EV. Три банки използват безплатен SSL/TLS сертификат.

2.2.4. Сравнителен анализ между централно европейските държави

Централно европейските държави, чиито банкови уеб сайтове изследваме, са три съседни държави без излаз на море, намиращи се в Централна Европа. Тези страни споделят много прилики в демографски и икономически план. От гледна точка на паричната политика, обаче, има някои съществени различия - Словакия е част от така наречената "Еврозона" и приема еврото за своя валута, а Чешката република и Унгария използват свои собствени валути. Банковите системи на последно споменатите две страни са формално независими от по-високо

равнище на надзор от страна на Европейската централна банка.

Таблица 2.2.1, таблица 2.2.4 и таблица 2.2.7 представят някои общи демографски, макроикономически и други данни за държавите, които имат общо значение в контекста на текущото проучване. Следва да се отбележи, че брутният национален доход (БНД, GNI), известен преди като брутен национален продукт (БНП), разработен от Световната банка, представлява общото вътрешно и чуждестранно производство на икономиката. Това е общото вътрешно и чуждестранно производство на страната, минус дохода, получен в националната икономика от нерезиденти. Показателят БНД на глава от населението се използва широко за оценка на общото равнище на икономическо развитие. Методът Атлас оценява размера на икономиките в щатски долари (World Bank, 2019). Коефициентът на Джини за доходното неравенство характеризира колко равномерно са разпределени благата сред жителите в държавата и представя в паричен вид разликата между благосъстоянието на бедните и богатите (Eurostat, 2019). Индексът за човешко развитие (HDI) е обобщен показател за постиженията в ключовите измерения на човешкото развитие: дълъг и здравословен живот, образование, равнището на доходите и др. (United Nations Development Programme, 2019).

Правителствата на всички три държави имат високи дългосрочни кредитни рейтинги от агенции за кредитен рейтинг като: Стандард енд Пуърс, Мудис и Фич. Кредитният рейтинг е оценка, направена от авторитетна агенция за кредитен рейтинг, и оценява кредитоспособността на страната. Използва се за оценка на вероятността дали правителството ще плати дълговете си. В това отношение той отразява международния имидж на правителствата (Country Economy, 2019).

От предоставените данни в таблица 2.2.1, таблица 2.2.4 и таблица 2.2.7 можем да подредим страните по отношение на цялостните

резултати на икономиките, както следва:

1. Чешката република е най-развита от трите страни, с най-висок БВП на глава от населението, БНД на глава от населението, HDI и кредитни рейтинги. Само индикаторът на Джини поставя страната на второ място.

2. Словакия се намира в средата по отношение на БВП на глава от населението, БНД на глава от населението, HDI и кредитни рейтинги. Само индикаторът на Джини поставя страната на последно място.

3. Унгария е най-слабо развита от трите държави с най-нисък БВП на глава от населението, БНД на глава от населението, HDI и кредитни рейтинги. Само индикаторът на Джини поставя страната на първо място.

Въз основа на данните, представени в таблица 2.2.3, таблица 2.2.6 и таблица 2.2.9, обобщихме резултатите от нашите изследвания в две таблици. В таблица 2.2.10 са дадени типовете SSL/TLS сертификати, използвани от чешките, словашките и унгарските банки. Таблица 2.2.11 представлява дяловете на издателите SSL/TLS сертификати на този пазарен сегмент.

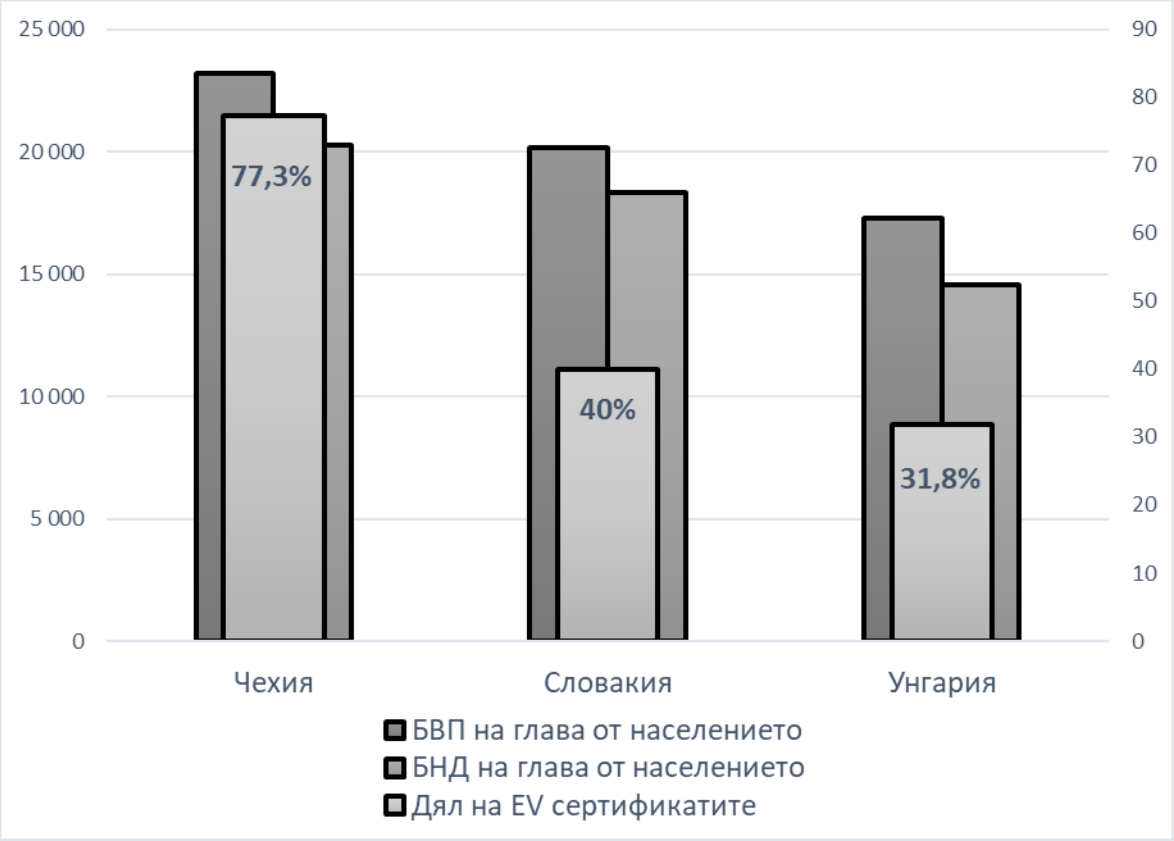
Таблица 2.2.10. Типове SSL/TLS сертификати, използвани в публичните сайтове на банки от централно европейските държави към август 2019 г.

Тип сертификат	**Чехия**		**Словакия**		**Унгария**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
Без сертификат	1	4,5	0	0	1	4,5
DV	4	18,2	9	60,0	14	63,6
EV	17	77,3	6	40,0	7	31,8
Общо	*22*	*100*	*15*	*100*	*22*	*100*

Таблица 2.2.11. Издатели на SSL/TLS сертификати, използвани в публичните сайтове на банки от централно европейските държави към август 2019 г.

Сертифициращ орган	**Чехия**		**Словакия**		**Унгария**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
-	1	4,5	-	-	1	4,5
Actalis	1	4,5	1	6,7	1	4,5
COMODO	1	4,5	1	6,7	-	-
DigiCert	7	31,8	4	26,7	5	22,7
Entrust	-	-	2	13,3		
GeoTrust	1	4,5	3	20,0	1	4,5
GlobalSign	-	-	-	-	1	4,5
Let's Encrypt	-	-	1	6,7	3	13,6
NetLock	-	-	-	-	7	31,8
RapidSSL	-	-	-	-	1	4,5
Sectigo	-	-	-	-	2	9,1
Thawte	11	50,0	3	20,0	-	-
Общо	*22*	*100*	*15*	*100*	*22*	*100*

Установихме, че в Чешката република по-голямата част от банките - 77,3%, използват EV сертификати; в Словакия – 40% от банките използват EV; и в Унгария – 31,8% от банките използват EV. На фигура 2.2.1 е разкрита връзката между показателите за икономическо развитие, като БВП на глава от населението и БНД на глава от населението, както и делът на сертификатите на EV SSL/TLS.



Фигура 2.2.1. Връзка между БВП/БНД на глава от населението и дела на SSL/TLS сертификатите

Забелязва се наличието на пряка положителна връзка между нивото на икономическо развитие и използването на по-сложните за издаване и по-скъпи сертификати EV. В Чешката република, която е най-икономически развита в сравнение с другите две страни, използването на сертификати за EV е най-високо. В Унгария, която е най-малко икономически развита спрямо другите две страни, използването на сертификатите EV е най-ниско. В Словакия, която е в средата на икономическото развитие между другите две страни, използването на сертификати EV също е на средно ниво.

Въз основа на данните в таблица 2.2.11 бихме могли да заключим, че има пряка отрицателна връзка между нивото на икономическо развитие и използването на безплатни SSL/TLS сертификати. В най-много икономически развита Чешка република, нито една от банките не използва безплатен SSL/TLS сертификат. В най-

малко икономически развита Унгария цели 3 банки използват безплатни SSL/TLS сертификати. В Словакия само 1 банка използва безплатни SSL/TLS сертификат.

Съществува голямо разнообразие от предпочитания за сертифициращия орган, но в Чешката република пазарът е най-консолидиран - 81,9% от банките използват 2 сертифициращи организации - Thawte (50%) и DigiCert (31,8%) като издатели на SSL/TLS сертификати. Ако отчетем, че компанията Thoma Bravo притежава DigiCert, Thawte и GeoTrust, то общият дял на групировката е 86,3%.

В Словакия пазарът е по-малко консолидиран - 66,7% от банките използват 3 сертифициращия органа: DigiCert (26,7%), Thawte (20%) и GeoTrust (20%) - и както вече споменахме всички те имат общ собственик.

В Унгария пазарът е най-малко консолидиран с най-голям брой сертифициращи органи. Странно е, че на този пазар Thawte не е представен. Вместо това вътрешният сертифициращ орган NetLock е лидер на пазара с 31,8% от издадените SSL/TLS сертификати. На второ място е DigiCert с 22,7% пазарен дял, а на трето място е Let's Encrypt с дял 13,6%.

Средният период на валидност на SSL/TLS сертификати за всяка държава е както следва: в Чехия е 20 месеца; в Словакия е 19 месеца; и в Унгария е 18 месеца. Тази разлика се дължи на 3-месечния срок на валидност на безплатните SSL/TLS сертификати.

Направеният сравнителен анализ ни дава основание да направим следните заключения:

Първо, пазарът на SSL/TLS сертификати в Чешката република е по-консолидиран, спрямо този в Словакия и Унгария.

Второ, що се отнася до използването на SSL/TLS сертификати, в Унгария има по-голямо разнообразие - 8 доставчици на SSL/TLS

сертификати, докато в Чешката република - са само 5, в Словакия - 7. В Чешката република най-популярния доставчик на SSL/TLS сертификати е Thawte с пазарен дял от 50% на уеб сайтовете на банките. В Словакия най-популярен е DigiCert с 26,7% дял. В Унгария най-популярният е вътрешният сертифициращ орган NetLock с 31,8% дял.

Трето, установи се пряка положителна връзка между показателите за икономическо развитие, като БВП на глава от населението и БНД на глава от населението, и вида на SSL/TLS сертификатите. Нивото на икономическо развитие влияе върху използването на по-сложните за издаване и по-скъпи сертификати EV. В Чешката република използването на сертификати EV е най-високо. В Словакия използването на сертификати EV е на средното ниво. В Унгария използването на сертификати EV е най-ниско. Подреждането в низходящ ред на трите страни по БВП на глава от населението и БНД на глава от населението, е същият - най-икономически развитата е Чешката република, следвана от Словакия и след това Унгария.

Четвърто, установи се пряка отрицателна връзка между нивото на икономическо развитие и използването на безплатни SSL/TLS сертификати. В най-икономически развитата Чешка република, никоя от банките не използва безплатни SSL/TLS сертификати. В Словакия 1 банка използва безплатен SSL/TLS сертификат. В най-малко икономически развитата Унгария 3 банки използват безплатни SSL/TLS сертификати.

Пето, поради краткия срок на валидност на безплатните SSL/TLS сертификати - 3 месеца, в по-слабо развитите страни средният период на валидност е по-малък. В Чешката република е 1 година и 8 месеца; в Словакия е 1 година и 7 месеца; и в Унгария е 1 година и 6 месеца.

Събраните данни се отнасят за определен период от време - към август 2019. Резултатите от проучването биха могли да имат важна практическа полза за управителите на банки и ИТ специалистите, когато

оценяват вариантите какви технологии да приложат, за да се сведат до минимум рисковете за финансовата институция. Също така, резултатите разкриват някои добри практики, използвани в чешките, словашките и унгарските банки. Изследванията, проведени върху използването на HTTPS протокола на публичните банкови уеб сайтове обхвана всички 22 чешки, 15 словашки и 22 унгарски банки, лицензирани да оперират на съответната територия под надзора на местните централни банки.

2.3. Изследване на банки от балканския полуостров

2.3.1. България

Целта на настоящото емпирично изследване е да се проучи какви технологии се използват в уеб сайтове на българските банки. Обхвата на изследването, представено по-долу, е проведено в периода юли 2016 г. - март 2019 г., и е ограничен до главния сайт на банката (Петров, 2016; Петров, 2017; Петров, 2019; Petrov et al., 2019a). Подобни изследвания, но с по-ограничен обхват са извършвани и в предходни периоди (Петров, 2010; Петров, 2013). При събирането на данните и анализа е използван експертен подход, т.е. употребата на автоматизирани средства е сведена до минимум с цел по-голяма достоверност на резултатите. Въпреки това, следва да се има в предвид, че е възможно част от данните, които връщат като отговор уеб сървърите, умишлено да са манипулирани от системните администратори, които ги поддържат, с цел по-голяма безопасност (Petrov & Dimitrov, 2019).

Таблица 2.3.1. Демографски, макроикономически и други показатели за България

Показател	**Стойност**
Парична единица	BGN
Население (2018) [милиони]	7
Площ [км^2]	110 994

БВП (номинален) на глава от населението (2018) [USD]	9 273
БНД на глава от населението (метод Atlas, 2017) [USD]	7 860
Индекс за доходно неравенство Gini (2018)	39,6
Индекс на човешкото развитие HDI (2017)	0,813 (много висок)
Рейтинг на S&P Global (дългосрочен, 2018)	BBB-
Рейтниг на Moody's Investors Service (дългосрочен, 2017)	Baa2
Рейтниг на Fitch (дългосрочен, 2019)	BBB

(Източници: : Country Economy, 2019; Eurostat, 2019; National Statistical Institute, 2018; UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Списъкът с банките, лицензирани да извършват дейност в Република България към средата на 2016 г. е получен от сайта на БНБ (таблица 2.3.2) и се състои от 22 банки (БНБ, 2016). Сайтовете на клоновете на чуждестранни банки (5 на брой и 1 в процес на заличаване от търговския регистър) не са включени в изследването.

Таблица 2.3.2. Използване на протокол HTTPS от български банки в периода юли 2016 - март 2019 г.

№	**Банка**	**Банков домейн**	**HTTPS**		
			07.2016	**11.2017**	**03.2019**
1	Алианц Банк България АД	allianz.bg	да	да	да

2	Банка ДСК ЕАД	dskbank.bg	да	проблеми	да
3	Банка Пиреос България АД	piraeusbank.bg	да	да	да
4	Българо-американска кредитна банка АД	bacb.bg	да	да	да
5	Българска банка за развитие АД	bbr.bg, bdbank.bg	НЕ	проблеми	проблеми
6	Инвестбанк АД	ibank.bg	НЕ	НЕ	да
7	Интернешънъл Асет Банк АД	iabank.bg	НЕ	НЕ	НЕ
8	Обединена българска банка АД	ubb.bg	да	да	да
9	Общинска банка АД	municipalbank.bg	НЕ	НЕ	да
10	ПроКредит Банк (България) ЕАД	procreditbank.bg	да	да	да
11	Първа инвестиционна банка АД	fibank.bg	да	да	да
12	Райфайзенбанк (България) ЕАД	rbb.bg	да	да	да
13	СИБАНК ЕАД	cibank.bg	да	да	-
14	Сосиете Женерал Експресбанк АД	sgeb.bg, expressbank.bg	да	да	да
15	Тексим Банк АД	teximbank.bg	НЕ	НЕ	да
16	Ти Би Ай Банк ЕАД	tbibank.bg	да	да	да
17	Токуда Банк АД	tcebank.com, tokudabank.bg	НЕ	проблеми	да

18	Търговска банка Виктория ЕАД	tbvictoria.bg	НЕ	проблеми	-
19	Търговска Банка Д АД	dbank.bg	НЕ	НЕ	да
20	УниКредит Булбанк АД	bulbank.bg, unicreditbulbank.bg	да	проблеми	да
21	Централна кооперативна банка АД	ccbank.bg	НЕ	проблеми	да
22	Юробанк България АД	postbank.bg	да	да	да

Информацията, която е събрана за сървърните технологии, е по отношение на вида на поддържания HTTP протокол, поддръжка на SSL/TLS и издател на сертификата, уеб сървър, използван език за програмиране, софтуерна платформа и местоположение на хостинга (таблица 2.3.3). В хода на изследването се установи, че никой от сървърите все още не поддържа HTTP/2, чиято спецификация е публикувана предходната година (Belshe et al., 2015).

Таблица 2.3.3. Основни технологии от страна на сървъра, издател на SSL/TLS сертификат (ако има) и местоположение на сървъра към юли 2016 г.

Банка №	**Версия на HTTP**	**Поддръжка на HTTPS и издател на сертификат**	**Уеб сървър**	**Програмен език**	**Платформа**	**Държава**
1	1.1	RapidSSL, GeoTrust	Apache	JSP	NETMIND	DE

2	1.1	Symantec	IIS/7.5	ASP.NET	Windows	BG
3	1.1	GlobalSign CloudSSL	-	-	Windows, ARR/2.5, Orchard	DE
4	1.1	Symantec	-	PHP/5.6.24	-	BG
5	1.1	-	Apache/2.2.15	PHP/5.3.3	CentOS, Joomla, studiox.bg	BG
6	1.1	-	Apache/2.2.16	PHP/5.2.17	Unix, WordPress 3.4.1	BG
7	1.1	-	Apache/2.4.6	PHP/5.5.3	Ubuntu, Joomla	BG
8	1.1	GeoTrust	Apache	-	-	BG
9	1.1	-	IIS/6.0	ASP.NET 4.0.30319	Windows	BG
10	1.1	Symantec	nginx	PHP/5.2.17	-	BG
11	1.1	COMODO	nginx	-	-	BG
12	1.1	Symantec	nginx/1.6.2	Python/ Django	-	BG
13	1.1	GlobalSign	Apache	-	-	BG
14	1.1	COMODO	NULL	PHP	Joomla, studiox.bg	BG
15	1.1	-	Apache	PHP	-	BG
16	1.1	GeoTrust	Apache	PHP/5.4.24	-	BG
17	1.1	-	Apache/2.2.9	PHP/5.2.6	Debian 5, Symfony	BG
18	1.1	-	Apache	PHP	-	BG
19	1.1	-	Apache/2.2.9	PHP/5.2.6	-	BG

20	1.0	RapidSSL, GeoTrust	-	-	-	BG
21	1.1	-	Apache/2.2.22	-	-	BG
22	1.1	GeoTrust	IIS/7.5	-	Windows	BG

Интерес представлява фактът, че 9 банкови сайта (41% от всички) не поддържат HTTPS и достъпът до главния сайт се извършва само по некриптирана връзка. Също 9 на брой, но други банкови сайта (41%) автоматично пренасочват посетителите към използване на HTTPS, ако първоначално се е използвал HTTP, което се счита за добра практика. С най-голяма популярност, сред 13-те банки, поддържащи HTTPS, като издател на SSL/TLS сертификати се ползва GeoTryst с дял от 38%, следван от Symantec с дял от 31%.

От софтуера за уеб сървъри най-използван е Apache - 55%, следван от IIS и nginx с 14%. Три от сървърите не дават информация за името и версията на уеб сървъра, но с много голяма степен на вероятност може да се предположи, че единият от тях е IIS, при което делът на последния може да нарасне до 18%.

Езикът за програмиране, използван за динамично генериране на съдържанието, най-често е PHP - 50%, като в пет от сайтовете са използвани готови системи за управление на съдържанието (Content Management System, CMS), а именно Joomla, Wordpress и Orchard. По отношение на използваните операционни системи не могат да се направят обосновани заключения, тъй като има много неизвестности. Два от сървърите са разположени в Германия, а останалите 20 са в България.

Клиентските технологии имат отношение предимно към браузъра. Полето X-Frame-Options дава възможност за защита от т.нар. Clickjacking атака (Ross&Gondrom, 2013). JavaScript библиотеките, които използват програмистите, са много на брой, но тук се установи един

своеобразен монопол на една от тях. По отношение на аналитичната софтуерна платформа, използвана за следене на ползваемостта на сайта, повечето банки използват повече от един подобен продукт (таблица 2.3.4).

Таблица 2.3.4. Стойност на полето X-Frame-Options в заглавната част на HTTP отговора (ако има), базови библиотеки на JavaScript и платформа за анализ на ползваемостта на уеб сайта към юли 2016 г.

Банка №	**X-Frame-Options**	**JavaScript библиотеки**	**Аналитични платформи**
1	SAMEORIGIN	Modernizr	GA
2	SAMEORIGIN	Flexcroll, jQuery-1.6.4, jQuery-UI-1.10.4, Knockout-3.2.0, MicrosoftAjax, Prototype, Scriptaculous, Sifr	FBR, GA
3	-	Bootstrap, Gdp-Data, jQuery-1.9.1,	GA, GTM
4	-	jQuery-1.11.3, Modernizr-2.8.3, Owl.Carousel2	DC, GAWCT, GA, GR
5	-	Imagesloaded, jQuery, Prefixfree, Sly.jQuery	GA
6	-	jQuery, jQuery-UI-1.8.18	GA
7	-	jQuery-1.10.1, Modernizr	GA
8	DENY	Modernizr-2.6.2	FBR, GTM
9	-	jQuery-1.7.2	GA
10	-	Cufon-Yui, jQuery, jQuery-UI-1.8.17	GA
11	-	jQuery-1.7.2, Prototype,	DC, FBR,

		Scriptaculous	GAWCT, GA, GR
12	SAMEORIGIN	jQuery	DC, FBR, GAWCT, GA, GR, GTM
13	SAMEORIGIN	jQuery-1.7.2, Thickbox, Webtoolkit.Url	GA
14	SAMEORIGIN	jQuery, jQuery-UI, Owl.Carousel, Rangeslider	FBR, GA, GTM
15	-	Cufon-Yui, jQuery-1.7, jQuery-UI-1.9.1, Modernizr	DC, FBR, GAWCT, GA, GR
16	-	Addthis_widget	GA
17	-	Colorbox, jQuery-1.10.2, Modernizr, Owl.Carousel	GA
18	-	EasySlider1.7, jQuery-1.3.2	Gemius, GA
19	-	jQuery, Prototype, Scriptaculous	DC
20	-	-	GA, GTM
21	-	jQuery	GA
22	-	jQuery-1.8.0	GA

Забележка: Съкращенията в последната колона са както следва: DC - DoubleClick.Net, FBR - Facebook Retargeting, GA - Google Analytics, GAWCT - Google AdWords Conversion Tracking, GTM - Google Tag Manager, GR - Google Remarketing

От събраните данни се установи, че библиотеката jQuery и/или jQuery-UI се използва в 82% от банковите уеб сайтове, Modernizr в 27% от тях. Фактът, че библиотеките Prototype и Scriptaculous се срещат винаги заедно, се дължи на това, че Scriptaculous използва Prototype, т.е.

последната дължи популярността си на Scriptaculous.

Много от сайтовете използват няколко платформи за аналитично следене на ползваемостта. Google Analytics - 91%, Facebook Retargeting - 27%, DoubleClick.Net - 23%, Google Tag Manager - 23%, Google AdWords Conversion Tracking - 18%, Google Remarketing - 18%.

Представените данни от емпиричното изследване може да послужат като ориентир за добрите практики в уеб технологиите в нашата страна. Разбира се, това наше твърдение не следва да се абсолютизира, тъй като са налице и някои аномалии - например липсата на осигуряване на криптирана връзка от 41% от изследваните сайтове. Въпреки, че не се обменя тайна информация между посетителя и уеб сайта, то подобна възможност не следва да се подценява от системните администратори.

По отношение на уеб сървъра се забелязва, че не се използват последните стабилни версии - към момента на изследването към средата на 2016 г. за Apache това е 2.4 (от февруари 2012), за IIS - 8.5 (от юли 2015), а за nginx - 1.10 (от май 2016). Последното може да се приеме за валидно само за тези уеб сървъри, които са давали информация, и то при това вярна, за версията.

Сред популярните езици за програмиране и уеб платформи са PHP, jQuery, Modernizr, Scriptaculous, Joomla. Само 27% имат защита срещу Clickjacking атаки чрез използване на полето X-Frame-Options.

Голяма част от сайтовете ползват по няколко платформи за аналитичен мониторинг - на Google, Facebook и DoubleClick.Net.

Изследването, проведено през юли 2016 г. беше повторено след 16 месеца - през ноември 2017 г. по най-важните показатели, отнасящи се до използването на HTTPS. Списъкът с банките - 22 на брой отново беше взет от сайта на БНБ (БНБ, 2017). В резултат, след събиране на данните, банковите уеб сайтове се групираха в три основни категории: уеб сайтове, използващи HTTPS без проблеми, уеб сайтове, използващи

HTTPS с проблеми и такива, неизползващи въобще HTTPS. По-характерното за всяка от тези категории е обобщено и анализирано по-долу.

Към категорията **банкови уеб сайтове, използващи HTTPS без проблеми към ноември 2017 г.**, спадат 50% от изследваните сайтове, а именно от таблица 2.3.2, тези с номера 1, 3, 4, 8, 10, 11, 12, 13, 14, 16 и 22 (таблица 2.3.5).

Таблица 2.3.5. Основни моменти при използване на протокол HTTPS от български банки към ноември 2017 г..

Банка №	**Автоматично пренасочване към HTTPS**	**Поддържане на HTTP/2**	**Тип на сертификат**	**Сертифициращ орган**	**Валидност в месеци**
1	да	не, HTTP/1.1	DV, wildcard	RapidSSL SHA256 CA	38
3	да	не, HTTP/1.1	EV	Symantec Class 3 EV SSL CA - G3	24
4	да	не, HTTP/1.1	EV	Symantec Class 3 EV SSL CA - G3	27
8	да	не, HTTP/1.1	DV	GeoTrust SSL CA - G3	38
10	не	не, HTTP/1.1	DV	Symantec Class 3 Secure Server CA - G4	38
11	да	не, HTTP/1.1	EV	COMODO RSA Extended Validation Secure Server CA	28
12	да	не, HTTP/1.1	DV	Symantec Class 3 Secure Server SHA256	26

				SSL CA	
13	да	не, HTTP/1.1	EV	GlobalSign Extended Validation CA - SHA256 - G3	13
14	не	не, HTTP/1.1	DV	COMODO RSA Domain Validation Secure Server CA	26
16	да	не, HTTP/1.1	EV	GeoTrust EV SSL CA - G4	25
22	да	не, HTTP/1.1	EV	GeoTrust EV SSL CA - G4	14

От представените данни се вижда, че около половината (5 банки) използват DV, а останалата част (6 банки) използват EV. По отношение на предпочитания за сертифициращ орган съществува голямо разнообразие, като с по-голяма популярност се ползват Symantec - 4 банки и GeoTrust - 4 банки (причисляваме RapidSSL към групата на GeoTrust, тъй като е тяхна собственост), а COMODO (2 банки) и GlobalSign (1 банка) са съответно по-малко предпочитани от банковите институции в нашата страна. Никоя от банките не използва HTTP/2.

В два от сайтовете на банките не се спазват добрите практики задължително да се извършва пренасочване от несигурна към сигурна връзка. При банка номер 14 допълнително бяха забелязани следните по-съществени проблеми: главната страница съдържа форма, която изпраща данни по HTTP, вместо по HTTPS (<form class="simple-form-search" method="get" action="http://www.sgeb.bg/bg/byrzi-vryzki/klonove-i-bankomati.html">). Прави впечатление, че в общо трите форми, които са на главната страница се използва различен подход за задаване на стойност на атрибута "action" - веднъж с указване на протокол HTTPS, веднъж без указване на протокол (т.е. относително адресиране по

протокол) и веднъж с указване на HTTP (<form action="https://www.sgeb.bg/bg/pages/search" method="get">, <form class="mobile-search" action="://www.sgeb.bg/bg/pages/search" method="get">). Това подсказва за липса на последователен подход при създаването на уеб страницата. Допълнителен проблем е, че се предлага за използване TLS 1.0, RSA и AES_128_CBC/HMAC-SHA1, които се считат за остаряло средство за реализиране на сигурна връзка. Тези проблеми не са съществени и поради това сайта на банката е оставен в тази категория.

Към категорията **банкови уеб сайтове, използващи HTTPS с проблеми към ноември 2017 г.**, спадат около 27% от изследваните сайтове, а именно от таблица 2.3.2, тези с номера 2, 5, 17, 18, 20 и 21 (таблица 2.3.6). Според нас това е сравнително голям процент, който показва подценяване на въпросите, свързани със сигурността на уеб сайтовете.

Таблица 2.3.6. Проблеми при използване на протокол HTTPS от български банки към ноември 2017 г..

Банка №	**Описание на проблема с HTTPS**
2	Код на грешката: SSL_ERROR_BAD_CERT_DOMAIN, Сертификатът е валиден само за dskbank.bg и https://www.dskbank.bg не е обхванат от него. Пренасочва несигурна връзка към сигурна връзка https://dskbank.bg/, тип на сертификата: EV, сертифициращ орган: Symantec Class 3 EV SSL CA - G3, период: 2 г., 2 м.
5	използване на самоподписан сертификат, изтекъл на 14.07.2012 г., предназначен за localhost.localdomain и показващ системна страница "Apache 2 Test Page powered by CentOS"
17	Код на грешката: SSL_ERROR_BAD_CERT_DOMAIN,

	ERR_CERT_COMMON_NAME_INVALID. Сертификатът е валиден само за следните имена: www.tokudabank.bg, tokudabank.bg. Предлага за използване TLS 1.0, RSA и AES_128_CBC/HMAC-SHA1, които се считат за остаряло средство за реализиране на сигурна връзка. Тип на сертификата: EV, сертифициращ орган: GeoTrust EV SSL CA - G4, период: 2 г., 2 м.
18	Код на грешка: SEC_ERROR_UNKNOWN_ISSUER, ERR_CERT_AUTHORITY_INVALID, самоподписан сертификат, издаден за невалиден домейн - citsyswebstntbv, пренасочва към незащитена версия на сайта (http://www.tbvictoria.bg/)
20	www.bulbank.bg - липсва запис в DNS, който да го свързва с IP адрес; http://bulbank.bg - пренасочва към https://www.unicreditbulbank.bg; https://bulbank.bg - Код на грешката: SEC_ERROR_UNKNOWN_ISSUER, ERR_CERT_AUTHORITY_INVALID, самоподписан сертификат, издаден за localhost, валиден 10 г.
21	Код на грешка: SSL_ERROR_RX_RECORD_TOO_LONG, ERR_SSL_PROTOCOL_ERROR. Порт 443 се използва, но приема само HTTP връзки, след което показва празна страница.

За систематизирането на данните в таблицата са използвани два различни браузъра - Mozilla Firefox 56.0 и Google Chrome 61.0 - последни версии към момента на извършване на проучването. Дадените кодове за грешки са върнати съответно от двата браузъра - първото съобщение е от Firefox, а второто - от Chrome. Проблемите варират в широки граници, като тези при сайтовете на банки с номера 2, 17 и 20 може да останат незабелязани от повечето потребители. За останалите сайтове на банки можем да дадем препоръката или да поддържат HTTPS

според добрите практики, или по-добре въобще да не използват HTTPS. Използването на самоподписани сертификати за localhost, неправилно конфигуриране на уеб сървъра за поддръжка на HTTPS и др. може да намали доверието в способностите на финансовата институция да поддържа системите си на съвременно ниво.

В случая с банка номер 2 може да се даде препоръка сертификата да обхваща както името на домейна без "www." отпред, така и с "www." отпред. Единствено поради тази причина сайта на банката е поставен в тази категория. Всички други показатели са в нормата.

В случая с банка номер 17 може да се даде препоръка за домейна www.tcebank.com да се използва отделен сертификат, различен от този на tokudabank.bg. Единствено поради тази причина сайта на банката е поставен в тази категория. Всички други показатели са в нормата.

Както и при другата група и в тази група не се поддържа протокол HTTP/2.

Към категорията **банкови уеб сайтове, неизползващи HTTPS към ноември 2017 г.**, спадат около 23% от изследваните сайтове, а именно от таблица 6.2, тези с номера 6, 7, 9, 15 и 19. Този висок процент, според нас, е тревожен факт. При положение, че цените за един DV сертификат започват от около 30 лв. за 1 година, единствената разумна причина да не се поддържа HTTPS е осигуряване на съвместимост със стари устройства. Разбира се, възможни са и други причини.

В обобщение на резултатите, извършеното изследване, проведено през ноември 2017 г. относно използването на протокол HTTPS в публичните сайтове на банките в България, обхвана сайтовете на всичките 22 български банки, лицензирани от БНБ. Точно половината - 50% от изследваните сайтове използват HTTPS без забелязани съществени проблеми. Около 27% от уеб сайтовете предлагат връзка чрез HTTPS, но с множество проблеми - самоподписани сертификати, неправилни имена на домейни, липса на съдържание, неправилно

конфигуриране и прочие. Около 23% от уеб сайтовете въобще не предлагат връзка чрез HTTPS. От сайтовете, поддържащи HTTPS без проблеми, около половината (5 банки) използват DV тип сертификат, а останалата част (6 банки) използват EV тип сертификат. С най-голяма популярност като сертифициращи органи се ползват Symantec - 4 банки и GeoTrust - 4 банки. Други сертифициращи органи са COMODO - 2 банки и GlobalSign - 1 банка. Никоя от банките все още не използва най-новия протокол HTTP/2.

Изследванията, проведени през юли 2016 г. и ноември 2017 г. бяха извършени отново през март 2019 г. по най-важните показатели, отнасящи се до използването на HTTPS, 32 месеца след първото и 16 месеца след второто изследване. Списъкът с банките отново беше взет от сайта на БНБ (БНБ, 2019). Броят на банките е 20 - с 2 по-малко от предходните изследвания, тъй като банките СИБАНК ЕАД (cibank.bg) и Търговска банка Виктория ЕАД (tbvictoria.bg) бяха заличени от регистъра на БНБ. В проучването са изключени уеб сайтове на чуждестранни банкови клонове и представителства на чуждестранни банки. Уеб сайтовете на банки, които оперират на трансгранична основа и уведомителен режим в рамките на ЕС също са изключени. Проучени са само местни банки, които работят под надзора на местната централна банка.

Основният метод, използван в проучването, включва анализ на отговорите, дадени от уеб сървърите. Google Chrome ver.73, работещ под типичен настолен компютър - Windows 10 Professional Edition x64, беше използван като уеб клиент с активиран модул "Developer Tools". Методиката на изследването се основава частично на методологията, използвана в предишни проучвания (Petrov, 2013; Petrov et al., 2017; Petrov et al., 2018a; Petrov et al., 2018b) върху уеб технологиите, използвани в банките. Основните характеристики при използването на HTTPS в публични уеб сайтове на български банки към март 2019 г. са

показани в таблица 2.3.7.

Таблица 2.3.7. Основни характеристики при използването на HTTPS в публични уеб сайтове на български банки към март 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	DV	RapidSSL RSA CA 2018	9
2	да	EV	DigiCert SHA2 Extended Validation Server CA	13
3	да	EV	DigiCert SHA2 Extended Validation Server CA	24
4	да	EV	COMODO RSA Extended Validation Secure Server CA	26
6	НЕ	DV	COMODO ECC Domain Validation Secure Server CA 2	6
8	да	EV	DigiCert SHA2 Extended Validation Server CA	24
9	да	EV	GeoTrust EV RSA CA 2018	24
10	НЕ	DV	DigiCert SHA2 Secure Server	24
11	да	EV	COMODO RSA Extended Validation Secure Server CA	26
12	да	EV	COMODO RSA Extended Validation Secure Server CA	24

14	да	DV	COMODO RSA Domain Validation Secure Server CA	24
15	да	EV	GeoTrust EV RSA CA 2018	24
16	да	EV	GeoTrust EV RSA CA 2018	24
17	да	EV	GeoTrust EV RSA CA 2018	9
19	да	EV	GeoTrust EV RSA CA 2018	14
20	да	DV	Let's Encrypt Authority X3	3
21	да	EV	COMODO RSA Extended Validation Secure Server CA	24
22	да	EV	GeoTrust EV RSA CA 2018	24

При анализа на данните, събрани през юли 2016 г., ноември 2017 г. и март 2019 г. се установи следното:

1. Делът на банките, използващи HTTPS без проблеми варира за разглеждания период: 59% към юли 2016 г., 50% към ноември 2017 г. и 90% към март 2019 г. Последните данни към март 2019 г. могат да се определят като силно положителни.

2. Делът на EV сертификатите расте: от 55% към ноември 2017 г. на 72% към март 2019 г.

2.3.2. Румъния

В хода на проучването, проведено през март 2019 г., бяха инспектирани главните уеб страници на 24 румънски банки. Основният метод, използван в проучването, включва анализ на отговорите, дадени от уеб сървърите. Google Chrome ver.73, работещ под типичен настолен компютър - Windows 10 Professional Edition x64, беше използван като уеб клиент с активиран модул "Developer Tools". Списъкът с банките, оторизирани да оперират в Румъния, е взет от уеб сайта на румънската

централна банка (Banca Naţională a României, 2019). В проучването са изключени уеб сайтове на чуждестранни банкови клонове и представителства на чуждестранни банки. Уеб сайтовете на банки, които оперират на трансгранична основа и уведомителен режим в рамките на ЕС също са изключени. Проучени са само местни банки, които работят под надзора на местната централна банка. В таблица 2.3.8 са представени данните за някои базови демографски, макроикономически и други показатели за Румъния. Използването на протокол HTTPS в публичните уеб сайтове на банки в Румъния към март 2019 г. е представено в таблица 2.3.9. Основните характеристики при използването на HTTPS в публични уеб сайтове на румънски банки към март 2019 г. са представени в таблица 2.3.10.

Таблица 2.3.8. Демографски, макроикономически и други показатели за Румъния

Показател	**Стойност**
Парична единица	RON
Население (2018) [милиони]	19,5
Площ [км^2]	238 397
БВП (номинален) на глава от населението (2018) [USD]	12 301
БНД на глава от населението (метод Atlas, 2017) [USD]	10 000
Индекс за доходно неравенство Gini (2018)	3511
Индекс на човешкото развитие HDI (2017)	0,811 (много висок)
Рейтинг на S&P Global (дългосрочен, 2014)	BBB-

Рейтниг на Moody's Investors Service (дългосрочен, 2018)	Baa3
Рейтниг на Fitch (дългосрочен, 2018)	BBB-

(Източници: Country Economy, 2019; Eurostat, 2019; National Institute of Statistics, 2018; UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Таблица 2.3.9. Използване на протокол HTTPS в публичните уеб сайтове на банки в Румъния към март 2019 г.

№	Банка	Банков домейн	HTTPS
1	Alpha Bank Romania S.A.	alphabank.ro	да
2	Banca Comerciala FEROVIARA S.A.	bfer.ro	проблем
3	Banca Comerciala Intesa Sanpaolo Romania S.A.	intesasanpaolobank.ro	да
4	Banca Comerciala Romana S.A.	bcr.ro	да
5	Banca de Export Import a Romaniei EXIMBANK S.A.	eximbank.ro	да
6	Banca Romana de Credite si Investitii SA	brci.ro	да
7	Banca Romaneasca S.A. Membra a Grupului National Bank of Greece	brom.ro, banca-romaneasca.ro	да
8	Banca Transilvania S.A.	bancatransilvania.ro	да
9	Bank Leumi Romania S.A.	leumi.ro	да
10	BRD - Groupe Societe Generale S.A.	brd.ro	да
11	CEC Bank S.A.	cec.ro	да
12	Credit Agricole Bank Romania S.A.	credit-agricole.ro	да
13	Credit Europe Bank (ROMANIA)	crediteurope.ro	да

	S.A.		
14	First Bank S.A.	firstbank.ro	да
15	Garanti Bank S.A.	garantibank.ro	да
16	Idea Bank S.A.	idea-bank.ro	да
17	Libra Internet Bank S.A.	librabank.ro	да
18	Marfin Bank (ROMANIA) S.A.	marfinbank.ro	да
19	OTP Bank Romania S.A.	otpbank.ro	да
20	Patria Bank S.A.	patriabank.ro	да
21	Porsche Bank Romania S.A.	porschebank.ro	да
22	ProCredit Bank S.A.	procreditbank.ro	да
23	Raiffeisen Bank SA	raiffeisen.ro	да
24	UniCredit Bank S.A.	unicredit.ro	да

Таблица 2.3.10. Основни характеристики при използването на HTTPS в публични уеб сайтове на румънски банки към март 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	EV	DigiCert SHA2 Extended Validation Server CA	14
3	да	EV	GeoTrust EV RSA CA 2018	26
4	да	EV	DigiCert SHA2 Extended Validation Server CA	12
5	да	DV	GeoTrust RSA CA 2018	26
6	да	EV	GeoTrust EV RSA CA 2018	14
7	да	EV	DigiCert SHA2 Extended Validation Server CA	5
8	да	EV	DigiCert SHA2 Extended Validation Server CA	13
9	да	EV	DigiCert Global CA G2	13

10	да	DV	Let's Encrypt Authority X3	3
11	да	EV	DigiCert SHA2 Extended Validation Server CA	14
12	да	EV	DigiCert Global CA G2	22
13	да	DV	Thawte RSA CA 2018	27
14	да	DV	GeoTrust RSA CA 2018	24
15	да	DV	DigiCert SHA2 Secure Server CA	12
16	да	DV	DigiCert SHA2 Secure Server CA	14
17	да	EV	DigiCert SHA2 Extended Validation Server CA	24
18	НЕ	EV	DigiCert Global CA G2	13
19	да	EV	GeoTrust EV RSA CA 2018	14
20	да	DV	COMODO RSA Domain Validation Secure Server CA	24
21	да	DV	Let's Encrypt Authority X3	3
22	да	DV	Go Daddy Secure Certificate Authority - G2	12
23	да	DV	DigiCert SHA2 Secure Server CA	26
24	да	EV	Actalis Extended Validation Server CA G1	12

В точка 4.4 данните са сравнени с тези от останалите балкански страни, обхванати в проучването.

2.3.3. Сърбия

В хода на проучването, проведено през март 2019 г., бяха

инспектирани главните уеб страници на 27 сръбски банки. Списъкът с банките, оторизирани да оперират в Сърбия, е взет от уеб сайта на сръбската централна банка (National Bank of Serbia, 2019). В проучването са изключени уеб сайтове на чуждестранни банкови клонове и представителства на чуждестранни банки. Уеб сайтовете на банки, които оперират на трансгранична основа и уведомителен режим в рамките на ЕС също са изключени. Проучени са само местни банки, които работят под надзора на местната централна банка. Основният метод, използван в проучването, включва анализ на отговорите, дадени от уеб сървърите. Google Chrome ver.73, работещ под типичен настолен компютър - Windows 10 Professional Edition x64, беше използван като уеб клиент с активиран модул "Developer Tools". В таблица 2.3.11 са представени данните за някои базови демографски, макроикономически и други показатели за Сърбия. Използването на протокол HTTPS в публичните уеб сайтове на банки в Сърбия към март 2019 г. е представено в таблица 2.3.12. Основните характеристики при използването на HTTPS в публични уеб сайтове на сръбските банки към март 2019 г. са представени в таблица 2.3.13.

Таблица 2.3.11. Демографски, макроикономически и други показатели за Сърбия към март 2019 г.

Показател	**Стойност**
Парична единица	RSD
Население (2018) [милиони]	7
Площ [км^2]	88 361
БВП (номинален) на глава от населението (2018) [USD]	7 234
БНД на глава от населението (метод Atlas, 2017) [USD]	5 180
Индекс за доходно неравенство Gini	37,8

(2018)	
Индекс на човешкото развитие HDI (2017)	0,787 (висок)
Рейтинг на S&P Global (дългосрочен, 2018)	BB
Рейтниг на Moody's Investors Service (дългосрочен, 2017)	Ba3
Рейтниг на Fitch (дългосрочен, 2018)	BB

(Източници: Country Economy, 2019; Eurostat, 2019; Statistical Office of the Republic of Serbia, 2017); UN HDRO, 2019; Wikipedia 2019; World Bank DRG, 2019)

Таблица 2.3.12. Използване на протокол HTTPS в публичните уеб сайтове на банки в Сърбия към март 2019 г.

№	Банка	Банков домейн	HTTPS
1	Addiko Bank AD Beograd	addiko.rs	да
2	Agroindustrijsko Komercijalna Banka AD, Beograd	aikbanka.rs	да
3	Banca Intesa AD Beograd (Novi Beograd)	bancaintesa.rs	да
4	Banka Postanska Stedionica AD, Beograd (Palilula)	posted.co.rs	да
5	Credit Agricole Banka Srbija AD Novi Sad	creditagricole.rs	да
6	Direktna Banka AD Kragujevac	direktnabanka.rs	да
7	Erste Bank AD, Novi Sad	erstebank.rs	да
8	Eurobank AD Beograd	eurobank.rs	да
9	Expobank AD Beograd	expobank.rs	проблем
10	Halkbank AD Beograd	halkbank.rs	НЕ

11	Jubmes Banka AD Beograd (Novi Beograd)	jubmes.rs	НЕ
12	Komercijalna Banka AD, Beograd (Vracar)	kombank.com	да
13	Mirabank AD Beograd-Novi Beograd	mirabankserbia.com	да
14	MTS Banka AD Beograd	mts-banka.rs	да
15	NLB Banka AD, Beograd	nlb.rs	да
16	Opportunity Banka AD, Novi Sad	obs.rs	да
17	OTP Banka Srbija AD, Novi Sad	otpbanka.rs	да
18	Piraeus Bank AD Beograd (Novi Beograd)	piraeusbank.rs, direktnabanka.rs	да
19	Procredit Bank AD, Beograd (Novi Beograd)	procreditbank.rs	да
20	Raiffeisen Banka AD Beograd	raiffeisenbank.rs	да
21	Sberbank Srbija A.D. Beograd	sberbank.rs	да
22	Societe Generale Banka Srbija AD, Beograd	societegenerale.rs	да
23	Srpska Banka AD Beograd (Savski Venac)	srpskabanka.rs	да
24	Telenor Banka AD Beograd (Novi Beograd)	telenorbanka.rs	да
25	Unicredit Bank Srbija A.D., Beograd (Stari Grad)	unicreditbank.rs	да
26	Vojvodanska Banka AD Novi Sad	voban.rs	да
27	VTB Banka AD Beograd	vtbbanka.rs, apibank.rs	да

Таблица 2.3.13. Основни характеристики при използването на HTTPS в публични уеб сайтове на сръбски банки към март 2019 г.

Банка №	**Автоматично пренасочване към HTTPS**	**Тип**	**Издател на SSL/TLS сертификат**	**Валидност в месеци**
1	да	EV	Thawte EV RSA CA 2018	11
2	да	DV	Go Daddy Secure Certificate Authority - G2	14
3	да	DV	Entrust Certification Authority - L1K	24
4	НЕ	DV	Thawte RSA CA 2018	14
5	да	EV	GeoTrust EV RSA CA 2018	12
6	да	DV	cPanel, Inc. Certification Authority	3
7	да	DV	DigiCert Global CA G2	12
8	да	DV	Go Daddy Secure Certificate Authority - G2	36
12	да	DV	Go Daddy Secure Certificate Authority - G2	37
13	да	DV	Thawte RSA CA 2018	13
14	да	DV	GlobalSign Organization Validation CA - SHA256 - G2	25
15	да	DV	COMODO RSA Domain Validation Secure Server CA	24
16	НЕ	DV	Let's Encrypt Authority X3	3
17	да	EV	GeoTrust EV RSA CA 2018	14
18	да	DV	cPanel, Inc. Certification	3

			Authority	
19	да	DV	Thawte RSA CA 2018	14
20	да	EV	Thawte EV RSA CA 2018	24
21	да	EV	GeoTrust EV RSA CA 2018	14
22	да	DV	Thawte RSA CA 2018	24
23	НЕ	DV	cPanel, Inc. Certification Authority	3
24	НЕ	DV	Let's Encrypt Authority X3	3
25	да	DV	Actalis Organization Validated Server CA G1	12
26	да	EV	GeoTrust EV RSA CA 2018	14
27	да	DV	cPanel, Inc. Certification Authority	3

В следващата точка данните са сравнени с тези от останалите балкански страни, обхванати при проучването.

2.3.4. Сравнителен анализ между балканските държави

В сравнителния анализ са включени банки, които са регулирани от местните централни банки и са от три съседни европейски държави, разположени на Балканския полуостров - България, Румъния и Сърбия. Тези страни имат много сходства в демографските и икономическите характеристики. От политическа гледна точка обаче има съществени разлики. България и Румъния са членки на ЕС и НАТО, докато Сърбия не е. От финансова гледна точка страните не използват еврото като своя валута и техните банкови системи са формално независими от надзора на Европейската централна банка.

Всички правителства имат високи дългосрочни кредитни рейтинги от агенции за кредитен рейтинг като Standard & Poor's, Moody's и Fitch Ratings.

Обобщените резултати от проучените банкови начални уеб страници са представени в няколко таблици въз основа на следните ключови показатели: наличие на автоматично пренасочване към HTTPS, тип сертификат, сертифициращ орган и период на валидност на SSL/TLS сертификата.

Интерес представлява фактът, че всички румънски банкови уеб сайта използват HTTPS (само една с проблем, но използва HTTPS), докато 5% от българските и 7,4% от сръбските банкови уеб сайтове изобщо не използват HTTPS.

Една българска, една румънска и една сръбска банка имат уеб сайтове, които използват HTTPS с проблеми. Българска банка №5 (от таблица 2.3.2) използва изтекъл през 2012 самоподписан сертификат за "localhost.localdomain". Румънска банка №2 (от таблица 2.3.9) използва сертификат, в който името на хоста не съвпада с главния домейн, а само с поддомейна bcfonline.bfer.ro. Сръбска банка №9 (от таблица 2.3.12) използва самоподписан сертификат за "marfin.gridsrv.net" с период на валидност от 10 години до 2027. В тези случаи може да се даде следната препоръка: или да се поддържа HTTPS според добрите практики, или по-добре да не се използва въобще HTTPS, тъй като тези проблеми биха могли да отслабят доверието на клиентите в способността на финансовата институция да поддържа своите информационни системи.

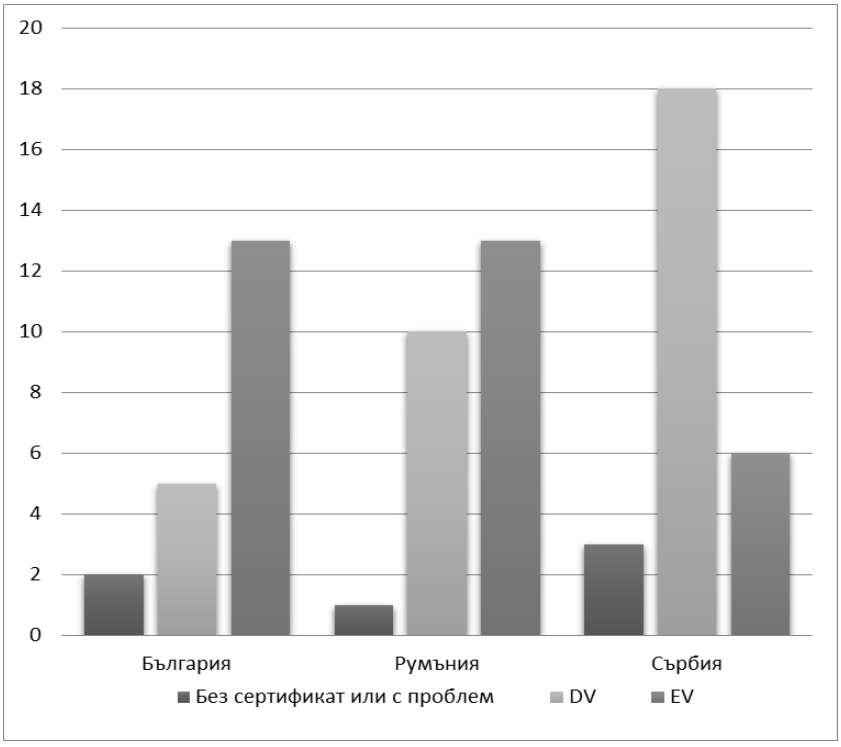
Около 90% от българските, 96% от румънските и 89% от сръбските банкови уеб сайта използват HTTPS без съществени проблеми. Те използват сертификати от признати сертифициращи органи, които удостоверяват връзката между публичния ключ, използван за шифроване на връзката и името на домейна, съхранявани на DNS сървъри.

От данните, представени в таблица 2.3.7, таблица 2.3.10 и таблица 2.3.13 се вижда, че в България мнозинството (13 банки) използва EV, а останалите (5 банки) използват DV. В Румъния

ситуацията е същата - мнозинството (13 банки) използва EV, а останалите (10 банки) използват DV. Но в Сърбия е обратното - мнозинството (18 банки) използват DV, а останалите (6 банки) използват EV (таблица 2.3.14, фигура 2.3.1).

Таблица 2.3.14. Типове SSL/TLS сертификати, използвани в публичните сайтове на банки от балканските държави към март 2019 г.

Тип сертификат	**България**		**Румъния**		**Сърбия**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
Без сертификат или с проблем	2	10	1	4	3	11
DV	5	25	10	42	18	67
EV	13	65	13	54	6	22
Общо	*20*	*100*	*24*	*100*	*27*	*100*



Фигура 2.3.1. Дялове на типовете сертификати в балканските държави

При два случая в България, един случай в Румъния и четири случая в Сърбия добрите практики не се следват от банковите уеб сайтове и те не пренасочват автоматично от несигурна HTTP към сигурна HTTPS връзка.

Таблица 2.3.15. Издатели на SSL/TLS сертификати, използвани в публичните сайтове на банки от балканските държави към март 2019 г.

Сертифициращ орган	**България**		**Румъния**		**Сърбия**	
	Брой	**%**	**Брой**	**%**	**Брой**	**%**
Actalis	-	-	1	4	1	4
COMODO	6	33	1	4	1	4
cPanel	-	-	-	-	4	17
DigiCert	4	22	12	52	1	4
Entrust	-	-	-	-	1	4
GeoTrust	6	33	5	22	4	17
GlobalSign	-	-	-	-	1	4
Go Daddy	-	-	1	4	3	13
Let's Encrypt	1	6	2	9	2	8
RapidSSL	1	6	-	-	-	-
Thawte	-	-	1	4	6	25
Общо	*18*	*100*	*23*	*100*	*24*	*100*

От обобщените данни, представени в таблица 2.3.15 се забелязва, че има голямо разнообразие от предпочитания за сертифициращ орган, особено в Сърбия. С най-голяма популярност в България се ползват: Comodo и GeoTrust - по 6 банки, DigiCert - 4 банки. Трябва да се отбележи, че RapidSSL е собственост на GeoTrust, така че всъщност истинският лидер на този пазарен сегмент е GeoTrust. В Румъния най-популярната сертифицираща организация е DigiCert - 12 банки,

последвана от GeoTrust - 5 банки. В Сърбия най-популярните сертифициращи организации са други участници на пазара, които не са представени в България и са слабо представени в Румъния: Thawte - 6 банки, cPanel - 4 банки, Go Daddy - 3 банки. Единственият голям общ играч е GeoTrust - 4 банки (таблица 2.3.15).

Една банка в България и по две банки в Румъния и Сърбия използват безплатни 3-месечни сертификати.

В резултат на проведеното сравнение, можем да направим следните по-съществени заключения:

Първо, банковият сектор в България е по-консолидиран спрямо този в Румъния и Сърбия.

Второ, що се отнася до използването на SSL/TLS сертификати, в Сърбия има повече разнообразие - десет издатели на SSL/TLS сертификати, докато в България те са само пет, а в Румъния - седем.

Трето, в Сърбия най-популярния доставчик на SSL/TLS сертификат е Thawte с дял от 25% уеб сайтове. В България най-популярният е GeoTrust с 39% дял (ако се включи и RapidSSL). В Румъния най-популярният избор е DigiCert с 52% дял. Интересно е, че една българска и две румънски и сръбски банки използват безплатни сертификати от "Let's Encrypt".

Четвърто, уеб сайтовете на две банки в България, една банка в Румъния и четири банки в Сърбия не пренасочат автоматично от незащитена HTTP връзка към защитена HTTPS връзка.

Пето, две банки в България, една банка в Румъния и три банки в Сърбия въобще не използват HTTPS или имат някакъв проблем със сертификатите.

Шесто, в България средната валидност на сертификатите е 19 месеца, при медиана - 24 месеца, докато в Румъния и Сърбия периодът на валидност е по-къс - 16 и 15 месеца, при медиана и при двете държави от 14 месеца.

Седмо, от банки, които използват HTTPS без проблеми, в България мнозинството (13 банки) използват по-сложните за издаване EV сертификати, а останалите (5 банки) използват по-обикновените DV сертификати. В Румъния ситуацията е подобна - мнозинството (13 банки) използва EV, а останалите (10 банки) използват DV. В Сърбия е обратното - мнозинството (18 банки) използва DV, а останалите (6 банки) използват EV. Предполагаме, че една от причините за това е, че в повечето случаи DV сертификата е по-евтин от EV сертификата.

Събраните данни са обвързани към точно определен период - март 2019 г. Резултатите от проучването биха могли да имат голяма практическа полза за управителите на банки и ИТ специалистите при оценяването на технологичните варианти, които трябва да се използват, за да се сведе до минимум рискът за финансовата институция. Също така резултатите разкриват някои добри практики, използвани в българските, румънските и сръбските банки. Изследванията, проведени с цел определяне използването на протокола HTTPS на публичните уеб сайтове на банките, обхващат сайтовете на всички 20 български, 24 румънски и 27 сръбски банки, лицензирани да оперират на територията на съответната държава от местните централни банки.

ГЛАВА ТРЕТА. ПОДХОДИ ЗА СЪХРАНЯВАНЕ НА ДАННИ ЗА ЕФЕКТИВНА ОБРАБОТКА В РЕАЛНО ВРЕМЕ

3.1. Съхраняване на данни от страна на уеб клиента

Динамиката на бизнес процесите в икономиката налага създаването на информационни системи с по-голяма степен на оперативност. Съответно използваните информационни технологии трябва да осигурят възможност за получаване, обработване и извличане на информация за обекта на изследване в реално време. Това е възможно да се реализира чрез използването на Интернет, съвременни програмни и комуникационни средства. Изграждането им, от друга страна, се определя от различни специфични изисквания и ограничения, като: изискване за работа в реално време или изискване за работа в режим "онлайн"[1], определено ниво на производителност, сигурност, безотказност, възможност за бъдещо развитие и надграждане на функционалността, мащабируемост, настройване, стандартизация, време за разработка, цена на разработка, цена на поддръжка и т.н.

При разработката на информационни системи важен момент е определяне на инструменталните средства за програмна реализация на потребителския интерфейс. Към настоящия момент на развитие на информационните технологии разработката на потребителския интерфейс на една информационна система се свежда до следните основни варианти:

а) десктоп приложение, използващо прозоречен интерфейс и работещо под графична операционна система от типа Windows, X

[1] В контекста на изложението, според нас, разликата между работа в реално време и работа в режим "онлайн" е, че при работа в реално време се изисква интервалът от време между момента на настъпване на дадено събитие до момента на получаване на данни за това от информационната система или субекта, извършващ изследването, да е пренебрежимо малък. Работа в реално време предполага работа в режим "онлайн", но обратното не винаги е вярно. Съществен проблем тук е дистанционната отдалеченост, при което се появява осезаемо времезабавяне при осъществяване на комуникация (Петров, 2015а).

Windows, Mac OS и други подобни;

б) уеб базирано приложение, работещо посредством използването на уеб браузър, който се свързва със сървърната част на разпределено клиент-сървър уеб приложение;

в) мобилно приложение, специално предназначено за използване на смартфон или таблет, работещо под операционна система Android, iOS и други подобни.

От гореизброените варианти най-универсален характер притежава подходът за разработка на информационна система с уеб базиран интерфейс, тъй като тя може да се използва от широк кръг устройства - както от настолни и преносими персонални компютри, така и от мобилни устройства - смартфони и таблети. Докато мобилните системи и десктоп системите могат да се използват само на определен тип хардуерни устройства със съответна специфична операционна система, то уеб технологиите са независими от вида на устройството и операционната система, работеща на заден план.

По-нататък в изложението спираме своето внимание само на информационните системи, работещи в реално време в уеб пространството. Основните проблеми, които разглеждаме, са:

1. Създаване на специализиран модел на информационната система - архитектура, в който точно да са дефинирани рамките и отговорностите на всеки компонент. За основа трябва да се използва архитектурата клиент-сървър.

2. Дефиниране и имплементиране на подход за обмен на данни в реално време между отделните компоненти на разпределеното клиент-сървър приложение.

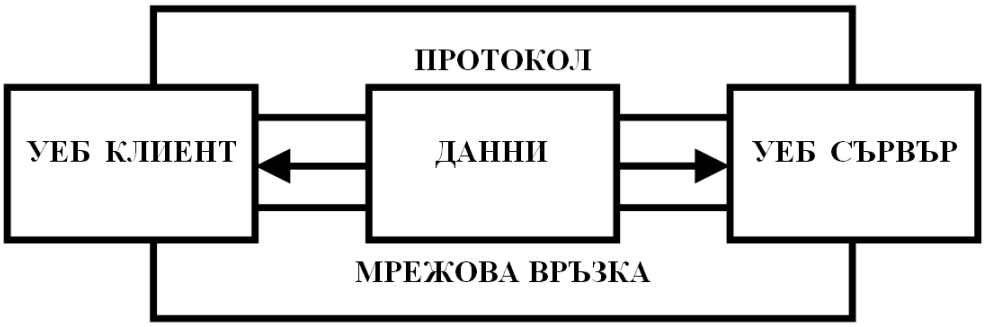
3. Подбиране на инструменталните средства, с помощта на които се изгражда сървър на приложения, който реализира програмната (бизнес) логика на информационната система.

4. Дефиниране и имплементиране на подход за съхраняване на

данни с оглед ефективната им обработка в реално време.

Освен гореизброените, може да се набележат и други проблеми, за решаването на които би могло да се приложат по-стандартни (класически) подходи и затова ги считаме за по-маловажни в рамките на настоящото изложение.

Архитектурата на информационна система за работа в реално време с използване на уеб технологии се базира на класическата клиент-сървър архитектура за създаване на разпределени уеб приложения (фигура 3.1). При нея уеб клиентът и уеб сървърът са различни приложения, които посредством мрежова връзка комуникират помежду си по правила, определени от протоколи от приложно ниво на модела OSI/ISO, както се посочва в стандарта ISO/IEC 7498-1:1994 (E). Мрежовата връзка се реализира на ниво операционна система, която с помощта на драйвъри използва протоколите от по-ниско ниво на модела OSI - протоколите TCP/IP.



Фигура 3.1. Базова архитектура уеб клиент - уеб сървър

В зависимост от това как се съхраняват и обработват данните от страна на сървъра, в теорията е прието клиент-сървър архитектурата да се разделя на три основни вида: двуслойна, трислойна и многослойна архитектура (Наков и колектив, 2005, с.673-677). При двуслойната клиент-сървър архитектура уеб клиентът и уеб сървърът комуникират само и единствено помежду си, без уеб сървърът да влиза в ролята на

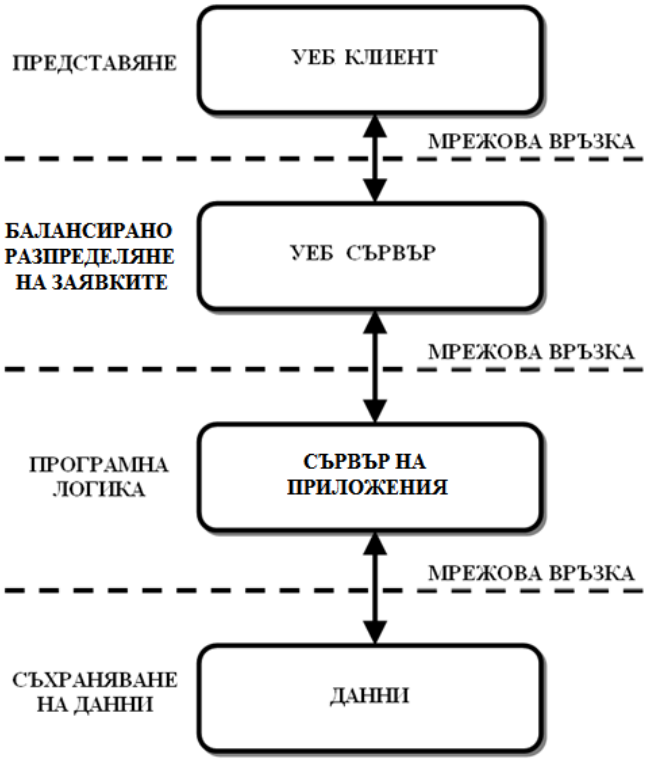
клиент и да се свързва с други сървъри. Данните се съхраняват или в оперативната памет, или в локалната файлова система. Възможно е да се използват двоични файлове, текстови файлове (CSV, TSV, XML, JSON и др. формати), бази от данни от вида DBM (например Berkeley DB на (Oracle, 2019)) или програмни библиотеки за реализация на релационни бази от данни без използването на самостоятелно сървърно приложение (например SQLite, Firebird Embedded (Firebird Foundation Inc., 2019) и др.).

Този подход е удачно да се използва при сравнително малки приложения и когато се работи едновременно с малък брой уеб клиенти, тъй като този вид клиент-сървър архитектура няма свойството добре да се мащабира. При по-голямо натоварване - едновременно нарастване на броя на клиентите и нарастване на броя заявки в секунда, хостът, на който работи уеб сървъра, се претоварва. За сравнение, при вида трислойна архитектура посредством мрежова връзка данните могат да се съхраняват на друг, специално предназначен за целта, хост.

При трислойната архитектура се добавя допълнително сървърно приложение, което да поеме функцията за съхраняване и извличане на данни. Това може да е под формата на специализиран файлов сървър, NoSQL сървър или сървър на система за управление на релационна база от данни (СУРБД). Изборът коя технология да се използва най-често е в зависимост от: сложността на приложението и на структурата на обработваните данни; необходимост от много високо бързодействие; и дали данните ще се споделят и с други приложения. Често в практиката широко се използва и комбиниран вариант за съхранение на данни, включващ и файлове, и NoSQL, и СУРБД.

Съвременен подход е използването на **многослойна клиент-сървър архитектура** (фигура 3.2), при който значително се изменя ролята на уеб сървъра. При нея нов момент е обособяването на нов слой - **сървър на приложения** (възможни са и други варианти - виж Cao et

al., 2013), който изолира приложенията, реализиращи програмно бизнес логиката, от директен контакт с потенциално опасната глобална мрежа и дава възможност уеб приложенията като цяло по-добре да се мащабират чрез изнасянето им на други хостове. По този начин разпределянето на натоварването от един на няколко хоста позволява да се създават приложения, които могат да обслужват едновременно много голям брой клиенти, отправящи много голям брой заявки в секунда.



Фигура 3.2. Многослойна клиент-сървър архитектура

Като използваме понятието "много голям брой" имаме предвид такива стойности, които биха превишили многократно нормалното натоварване на един хост. За нормално натоварване може да се приеме примерно средното натоварване на процесора, получено от командата top под Linux/Unix, което е по-малко от 1 за произволен интервал от време[1]. Стойности над 1 може да се приемат за голямо натоварване, а с един порядък повече - за много голямо натоварване. При използване на

[1] Самата команда връща средното натоварване за последните 1, 5 и 15 минути. За повече информация как се извършват изчисленията виж: (Walker, 2006).

показателя средно натоварване, той трябва да се раздели на броя на процесорите умножено по броя на ядрата на процесорите, т.е. дадената граница до 1 и над 1 е за еднопроцесорна система с едноядрен процесор.

При многослойната клиент-сървър архитектура в последните два слоя - слоя със сървъра на приложения и слоя със съхраняването на данни, са концентрирани по-голямата част от технологичните иновации, свързани с работа в реално време на сървърната страна на разпределеното уеб приложение. На предпоследния слой през последните години, по наша субективна оценка, следните сървъри на приложения се развиват интензивно и са перспективни за използване[1]: WildFly (старо име JBoss) за Java, appserver.io за PHP, Dancer за Perl и не на последно място по значение - Node.js.

На последния слой от многослойната клиент-сървър архитектура - слоя със съхраняването на данни, отново има много варианти за избор. Съхраняването на данни във файлове е подходящо при малки обеми данни и когато не се налага често обновяване. Ако се използват двоични файлове (копие на данните от оперативната памет), бързодействието е много по-голямо.

От друга страна, текстовите файлове са по-удобни, тъй като с тях лесно могат да работят и други приложения или човек ръчно да коригира данни с текстов редактор. Лесно се преобразуват и в различни формати. Технологичното развитие тук е, че в последните години се наложиха текстовите файлове във формати XML и JSON. Популярност в практиката набира и използването на т.нар. NoSQL системи (Panayotova, G. et al., 2016). Те биват най-различни видове, но общото между тях е, че предлагат много ефективно съхраняване, търсене и обработка на данни, които могат да се представят като двойки ключ-стойност. Някои

[1] Спираме своето внимание само на такива с лиценз свободен софтуер, без да омаловажаваме останалите.

системи използват само оперативната памет за съхраняване на данните и са подходящи за ролята на кеш и междинно съхраняване на данни между отделни заявки, а други дават възможност да се обработват огромни обеми от данни, разпределени на множество външни запаметяващи устройства.

Постоянното и сесийно съхраняване на данни при приложенията в реално време в зависимост от поставените цели може да се извършва както от страна на уеб клиента, така и от страна на уеб сървъра (Петров & Начева, 2019). През последните години се наложи нов подход за организация, съхраняване и достъп до данните при този тип приложения, а именно все повече данни да се съхраняват за по-кратки или за по-дълги периоди от време от страна на уеб клиента - това представлява временно кеширане на данни при клиента с цел избягване изпращането на едни и същи данни многократно от сървъра на клиента.

От страна на уеб клиента един от първите варианти за съхраняване на данни в исторически план е чрез т.нар. бисквитки (cookies) - данни, съхранявани в текстови файлове (Barth, 2011). Основен недостатък на бисквитките като средство за съхраняване на данни от гледна точка на обема на трафика е, че те се изпращат при всяка заявка, което увеличава обема на трафика, както и това че различните браузъри имат различни ограничения за обема бисквитки, които съхраняват. В HTML5 са въведени нови технологични средства за съхраняване на данни - т.нар. Web Storage обекти (Hickson, 2016), които биват два вида - за постоянно и за временно съхраняване на данни в браузъра[1].

При този подход с използване на Web Storage обекти данните отново се асоциират с определен URL адрес и могат да се използват само от програми, заредени от него. С тях се работи само чрез програми,

[1] Стандартът за Web Storage дефинира два вградени обекта: localStorage - за постоянно съхраняване, дори и при затваряне на браузъра; и sessionStorage - за временно (сесийно) съхраняване, изтрива се при затваряне на браузъра. И двата обекта имат методи .setItem('key', 'value') и .getItem('key') съответно за записване и за извличане на съхранена стойност.

написани на JavaScript, тъй като не се изпращат автоматично при всяка заявка както бисквитките. Обемът на съхраняваните данни, според препоръките на стандарта, са поне 5 MB на URL адрес и за разлика от бисквитките стойностите могат да са от произволен тип данни, а не само низове.

Съхраняването на данни от уеб клиент с използване на Web Storage обекти се осъществява по следния сценарии: когато уеб сървърът връща отговор, в заглавната му част на ред започващ със "Set-Cookie:" се изпращат данни, които уеб клиентът съхранява, като ги асоциира с URL адреса на ресурса. При следващи заявки до същия URL адрес уеб клиентът изпраща всички асоциирани с адреса данни в заглавната част на заявката на ред започващ с "Cookie:". Данните се състоят от двойки ключ-стойност и може да се използват чрез програми на JavaScript по следния начин:

```
//запис на бисквитка, която ще се изтрие след затваряне на уеб клиента
document.cookie = "user=ivan@example.com";
//запис на бисквитка, която ще се изтрие след зададена дата
document.cookie = "user=ivan@example.com; Expires=Thu, 14 May 2020 12:30:01
GMT;";
//изтриване на бисквитка - задаване на празна стойност и дата в миналото
document.cookie = "user=; Expires=Thu, 01 Jan 1970 00:00:00 GMT";
//извличане на всички данни, като низ "КЛЮЧ1:СТОЙНОСТ;
КЛЮЧ2:СТОЙНОСТ..."
var x = document.cookie;
//Обекти localStorage и sessionStorage:
//Постоянно съхраняване, дори и при затваряне на браузъра
localStorage.setItem('key', 'value');
//Извличане на стойност съхранена за постоянно
x = localStorage.getItem('key');
// Временно (сесийно) съхраняване, изтрива се при затваряне на браузъра
```

```
sessionStorage.setItem('key', 'value');
 // Извличане на стойност съхранена временно
x = sessionStorage.getItem('key');
// Извежда колко двойки има записани
alert(localStorage.length + "/" + sessionStorage.length);
```

Налични са и варианти за съхраняване на данни при уеб клиента от тип бази от данни, но не всички браузъри ги поддържат и работата по тяхната стандартизация продължава. Такива са Indexed Database API (Alabbas&Bell, 2017) и Web SQL Database (Hickson, 2010).

Работата в реално време изисква междинното съхраняване на данни в универсални формати с цел не само бърза употреба, но и лесна модификация на използваната информационна система при необходимост от включване на нови функционални възможности. Допълнително, съхраняването на данните и техния обмен в универсални платформено независими формати дава възможност да се използват различни инструментални средства за тяхната обработка, визуализиране и анализ. Съществуват множество на брой отворени стандарти за форматиране на данните и подходът за избор на формат за обмен на данни обикновено зависи от редица фактори. Можем да очертаем три най-често срещани варианти:

1) При ново разработени клиентска и сървърна част на разпределеното уеб приложение обикновено изборът се прави или на база изрични изисквания в заданието, или на база модерна към момента технология (следване на "модните тенденции"), или на база субективния опит на екипа, разработващ приложението.

2) При ново разработване само на клиентската или само на сървърната част, обикновено изборът се прави на база какви формати за обмен на данни поддържа отсрещната страна. Тук по-разпространен е вариантът, при който вече съществува сървърно приложение, предлагано като уеб услуга, а тепърва се разработва клиентската част.

Много често се предлага и API на различни езици за програмиране, при което няма възможност за избор на стандарт за обмен на данни.

3) При съществуващи клиент-сървър приложения, които вече имат определена широко използвана функционалност, очевидно става въпрос за надграждане, при вече използван формат за обмен на данни. В този случай следва да се използва съществуващия формат, тъй като смяната му може да доведе до необходимост от пренаписване на голяма част от програмния код и на клиентското и на сървърното приложение, което е свързано със значителни разходи.

От по-горе изложеното се вижда, че изборът на формат за обмен на данни е решение, което съществено се отразява на бъдещото развитие на едно разпределено приложение и изборът трябва да бъде добре мотивиран. Към настоящия момент широко се използват в практиката два текстови отворени формата за обмен и съхраняване на данни при уеб приложенията, които също са подходящи и за работа в режим реално време - форматите XML и JSON.

Форматът XML има универсален характер и съответно е "по-тромав" (т.е. по-обемен и по-бавен за обработка) спрямо JSON (Maeda, 2012; Goyal et al., 2017). Трябва да се отчита, че при приложенията в реално време асинхронните заявки към уеб сървъра от страна на клиента трябва да са малки по обем, бързо да се обработват и обикновено се извършват от програма написана на JavaScript. Ако от страна и на сървъра, и на клиента има приложения на JavaScript, по-голямо удобство на работа дава JSON.

Форматът **XML**[1] (съкратено от eXtensible Markup Language) представлява маркиращ език, който представлява опростена форма на ISO стандарта за маркиращ език Standard Generalized Markup Language (SGML)[2]. При използване на маркиращи езици, посредством специални

[1] Спецификацията е публикувана в (Bray et al., 2006).
[2] Описан е в "ISO 8879:1986 Information processing - Text and office systems - Standard Generalized Markup Language (SGML)".

символни комбинации, наречени маркери, се обозначават отделни части от текст като определен тип данни. Маркерите се наричат още тагове или етикети и на практика с тях се описват структурирани и йерархично подредени множества от данни. Маркираните с тагове данни се наричат елементи.

По синтаксис таговете, използвани в XML документите, приличат на тези, използвани в HTML уеб страниците, но с тази разлика, че таговете в HTML са предварително дефинирани да имат точно определен смисъл, докато тези в XML се дефинират от потребителя. Всъщност, ако една уеб страница е съставена по правилата на XHTML (Baker, 2010), тя представлява и валиден XML документ. Основните цели, залегнали при създаване на XML от страна на W3C, са били: да бъде лесен за използване през Интернет; да се поддържа от широк кръг приложения; да е съвместим с SGML; лесно да се пишат програми, които обработват XML документи; XML документите да са четими от хора и смислени по структура; дизайнът на XML да се изработва лесно и прочие. Организацията на XML документ е дадена в примера, който следва:

```
<?xml version="1.0" encoding="UTF-8"?>
<tweet id="900719925474099921">
    <likes_count>111</likes_count>
    <user>
        <screen_name>realDonaldTrump</screen_name>
        <followers_count>56800570</followers_count>
    </user>
<!-- КОМЕНТАР -->
    <place country="Bulgaria" place_type="city" />
</tweet>
```

Както се вижда, елементите са йерархично организирани и един елемент може да има родител (родителят на <user> е <tweet>) и деца

(децата на <user> са <screen_name> и <followers_count>). Атрибутите, които се задават на таговете, могат да се преобразуват като деца и обратно. Например тагът за празен елемент <place ... /> може да се преобразува така:

```
<place>
<country>Bulgaria</country>
<place_type>city</place_type>
</place>
```

Отварящият и затварящ таг <user>...</user> с елементи деца може да се преобразува като таг за празен елемент с атрибути по следния начин:

```
<user screen_name="realDonaldTrump " followers_count="56800570" />
```

Подобно на HTML, символите "<", ">", "&", апострофът и кавичките са запазени символи в езика и ако трябва да се използват, те трябва да се представят като < (за <), > (за >), & (за &), ' (за ') и " (за ").

Ако в данните се срещат много подобни символи, това преобразуване е неудобно и може да се използват т.нар. CDATA секции. В тези секции запазените символи може да се използват, без да се налага някакво специално кодиране. CDATA секция започва с "<![CDATA[" и завършва с "]]>". Съответно тези последователности от символи не трябва да се срещат в така маркираните данни.

Пример за секция CDATA:

```
<![CDATA[
2 + 3 < 10 < 5 + 7; true && false = false
]]>
```

Въпреки че XML документът на практика представлява обикновен текстов файл и при програмна обработка биха могли да се използват функции за обработка на символни низове, то обработката на XML документите е добре да се извършва от специализирани програмни

библиотеки, известни в практиката като "XML процесор" (El-Hassan et al., 2009). Това е софтуерен модул или самостоятелно приложение, който се използва за четене, създаване и редактиране на XML документи и чрез който се предоставят възможности за лесно боравене с тяхното съдържание и структура.

Правилата за синтаксиса на XML са сравнително елементарни, подобни са на тези на HTML и спазването им е задължително, с цел XML документът да бъде структуриран правилно и XML процесорите да могат да го обработват. Елементите са йерархично организирани и един елемент може да има "родител" и "деца". Атрибутите, които се задават на таговете, могат да се преобразуват като "деца" и обратно.

От страна на уеб клиента чрез използване на вградения обект DOMParser и метода .parseFromString(), може да се обработи символен низ и да се създаде в паметта йерархична структура, състояща се от елементи (Wenzel&Meinel, 2015; MDN Web Docs, 2018). За достъп до отделните елементи, техните стойности и атрибути се използва комбинация от различни средства: .documentElement - съдържа върха на йерархията (корена); .getElementsByTagName() - връща масив от елементи, отговарящи на зададения таг; .childNodes - съдържа масив от елементите деца спрямо текущия елемент; .parentNode - съдържа родителския елемент спрямо текущия елемент; .getAttributeNode() - връща масив от атрибути; .nodeValue - съдържа стойността на атрибута; .nodeName - съдържа името на тага; .attributes - съдържа масив от атрибутите на елемента.

Обработка на XML от страна на уеб клиента с помощта на вградения обект DOMParser е даден в примера, който следва.

Файл xml.html:

```
<script>
var s = ' \
    <tweet id="90071992547409921"> \
```

```
            <likes_count>111</likes_count> \
            <user> \
                <screen_name>realDonaldTrump</screen_name> \
                <followers_count>56800570</followers_count> \
            </user> \
            <place country="Bulgaria" place_type="city" /> \
        <tweet> \
';

var x = (new DOMParser).parseFromString(s, 'text/xml');
document.writeln("<pre>");

document.writeln("NODES: " + x.getElementsByTagName("*").length );
document.writeln("ROOT NODE: " + x.documentElement.nodeName );
document.writeln("id: " +
x.getElementsByTagName("tweet")[0].getAttributeNode("id").nodeValue );
document.writeln("likes_count: " +
x.getElementsByTagName("likes_count")[0].childNodes[0].nodeValue );
document.writeln(x.getElementsByTagName("user")[0].childNodes[1].nodeName + ": "
+
    x.getElementsByTagName("user")[0].childNodes[1].childNodes[0].nodeValue
);
document.writeln("followers_count: " +
x.getElementsByTagName("user")[0].getElementsByTagName("followers_count")[0].c
hildNodes[0].nodeValue
);
document.writeln("country: " +
x.getElementsByTagName("place")[0].getAttributeNode("country").nodeValue );
document.writeln(x.getElementsByTagName("place")[0].attributes[1].nodeName + ": "
+ x.getElementsByTagName("place")[0].attributes[1].nodeValue );
```

```
</script>
```

Резултат:

```
NODES: 11
ROOT NODE: tweet
id: 90071992547409921
likes_count: 111
screen_name: realDonaldTrump
followers_count: 56800570
country: Bulgaria
place_type: city
```

В примера в променливата "s" има данни във формат XML. Тези данни може да са получени чрез AJAX заявка, чрез WebSocket комуникация или по някакъв друг начин, и се третират като XML документ. Създава се нов обект x, на базата на вградения обект DOMParser и се извиква метода .parseFromString(), който обработва символния низ от променливата s и създава в паметта йерархична структура, състояща се от множество елементи.

От страна на уеб сървъра при Node.js има множество от модули за работа с XML. Най-популярните са: xml2js (Npmjs.org, 2017d), htmlparser2 (Npmjs.org, 2017c), sax (Npmjs.org, 2017b), libxmljs (Npmjs.org, 2018a) и др. За работа по сходен начин на гореспоменатия обект DOMParser, може да използва модула xmldom (Npmjs.org, 2017a).

3.2. Съхраняване на данни от страна на уеб сървъра

Системите от тип **NoSQL** са много подходящи за изграждане на уеб приложения, работещи в реално време. Те осигуряват много висока скорост при търсене, извличане и запис на данни, понякога в пъти по-висока от тази, предлагана от системите за релационни бази от данни (Петров, 2013). Това е една от причините, поради която се използват и за съхраняване и обработка на големи обеми от данни (Bhogal&Choksi,

2015; Zafar et al., 2016) в порядъка от десетки петабайта до няколко ексабайта[1].

Сред най-популярните[2] NoSQL системи е Redis (Redis.io, 2019), чиято разработка е била финансирана от компанията VMware в началните етапи. **Redis (REmote DIctionary Server)** е сървърна система от тип свободен софтуер, съхраняваща сложни структури от данни в оперативната памет. Като стойности могат да се съхраняват не само единични (скаларни) стойности, като числа и низове, но и хешове, списъци, множества, сортирани множества, масив от битове и др.

При работа с Redis за операционна система се препоръчва да се използва версия на UNIX/Linux (Redis.io, 2019), но има и различни прекомпилирани варианти за Windows, които могат да се ползват при разработка на приложения. Под Windows не е необходимо класическо инсталиране, тъй като системата е от типа "portable apps".

Ще посочим накратко най-характерните особености и команди на Redis, за да се изяснят по-подробно възможностите му. Полезни административни команди, които могат да се използват в програмата клиент (redis-cli.exe) са: PING - проверка на връзката със сървъра, при което той трябва да отговори с PONG (ако всичко е нормално, разбира се); INFO - извежда информация за настройките и статистики за сървъра; SELECT НОМЕР_НА_БД - задава коя да е активната база от данни. По подразбиране е тази с номер 0. Обикновено има 16 бази от данни; DBSIZE - връща броя на ключовете в текущата база от данни; CONFIG GET DATABASES - връща броя на базите от данни; FLUSHALL - изтрива всички ключове от всички бази от данни; FLUSHDB - изтрива всички ключове от активната база от данни; MONITOR - показва в реално време всички заявки към сървъра. Добре е

[1] 1 EB = 1000 PB = 1 млн. TB = 1 млрд. GB

[2] Според авторитетно изследване на австрийската консултантска фирма solidIT към началото на 2018 г, най-популярната NoSQL система от тип ключ-стойност е Redis: (DB-Engines, 2017a).

да се ползва с клиент в отделен прозорец.

Максималният размер на ключа е до 512 MB. Същото се отнася и за размера на стойността (Suri et al., 2018). Ако се налага да се работи с данни по-големи от 512 MB, те трябва да се разположат в няколко ключа. С характерните за хешовете команди SET, GET, DEL и EXISTS се манипулират данните. За разлика от работата с хешове, с командите MSET и MGET се работи с множество двойки ключ-стойност. Друга особеност е, че с командите INCR, INCRBY, DECR и DECRBY[1] към числа могат да се прибавят и изваждат други числа.

Както вече споменахме по-напред, освен единични стойности в базата от данни могат да се съхраняват и структури от данни. При работа с тези типове данни се използват както команди, подобни на дадените по-горе, така и някой нови команди, специфични за типа данни. Пред името на командата има буква, която показва за какъв тип данни се отнася: H - за хеш (Hash), L - за списък (List), S - за множество (Set), Z - за сортирано множество (sorted set) и др.

Специфичното при **хеша** (асоциативния масив), като стойност в Redis е, че и ключът, и стойността са низове. За да се разграничават ключовете, които са в стойността, от основния ключ, към който са асоциирани, те се наричат полета. За работа с единични двойки полета-стойност се използват командите: HSET, HSETNX, HGET, HDEL, HEXISTS, HINCRBY, HINCRBYFLOAT. За работа с множество двойки се използват: HMSET и HMGET. Нови команди са: HLEN - връща броя на двойките поле-стойност; HKEYS - връща всички полета от хеша; HVALS - връща всички стойности от хеша; HGETALL - връща всички двойки поле-стойност.

Списъците, като структура от данни в Redis, представляват

[1] По-долу няма да пояснява смисъла на командите. От дадените разяснения преди всяка група команди, след прочитане на името на командата (което в повечето случаи е мнемонично) може да се разбере какъв е нейният смисъл, т.е. името е самоописателно за специалисти работили с хешове. Там където смисълът не може да се схване от контекста, са дадени допълнителни пояснения.

подредено множество от елементи (в общия случай низове), всеки от който има пореден номер в списъка - индекс. Операциите със списък са същите, както операциите с динамичен масив (в някои езици за програмиране класове Vector и ArrayList). За вмъкване на елементи в списък съществуват различни варианти: LSET, LINSERT, RPUSH/LPUSH, RPUSHX/LPUSHX - добавя елемент в списъка само ако зададения ключ съществува, LLEN. За извличане на елементи от списък се използват командите: LINDEX (няма команда LGET), LRANGE, RPOP/LPOP. За премахване на елементи от списък се използват командите: LREM, LTRIM - премахва от списък елементи извън зададен затворен интервал, т.е. остават само елементите в този интервал.

Особен интерес представлява операция прилагана върху два списъка: RPOPLPUSH - маха последния елемент от един списък и го добавя в началото на друг списък.

Множествата, като типове данни в Redis, са подобни на списъците, с изключение на това, че съдържат уникални елементи. Освен това множествата нямат индекс и подредба. Реализират по програмен път понятието за множество в математиката. Основните операции са: SADD, SREM - премахва елементи от множество, SPOP - премахва и връща случаен елемент, SRANDMEMBER връща елементи избрани случайно от множеството, SISMEMBER, SMEMBERS, SCARD - връща броя на елементите.

Операции от теорията на множествата, включващи работа с няколко множества едновременно са: SMOVE - премества елемент от едно множеството в друго множеството; SUNION - връща обединение от множества, т.е. всички елементи и от едното и от другото множество, но без повторение - $A \cup B = \{x \mid x \in A \vee x \in B\}$; SUNIONSTORE - подобно на SUNION, но записва резултата в ново множество; SINTER - връща сечение от множества, т.е. елементите, които се срещат едновременно във всички множества - $A \cap B = \{x \mid x \in A \wedge x \in B\}$; SINTERSTORE - подобно

на SINTER, но записва резултата в ново множество; SDIFF - връща разлика (допълнение) на множества, т.е. елементите, които са част от първото множество, но не са част от второто множество. Операцията се прилага последователно върху двойки ключове, като междинният резултат участва като първо множество в следващата операция изваждане. $A - B = A \cap \overline{B} = \{x \mid x \in A \wedge x \notin B\}$; SDIFFSTORE - подобно на SDIFF, но записва резултата в ново множество.

От програмна гледна точка, интерес представляват **сортираните множества**, които са подобни на множествата, тъй като се състоят от уникални елементи. Подобни са и на списъците, тъй като елементите са подредени - на всеки елемент се задава реално число (score), което се използва за подреждане на елементите. Това число често се нарича тегло или приоритет в зависимост от контекста. Основните и специфични операции със сортирани множества, свързани с боравенето с елементи и техните тегла са: ADD - добавя елементи с техните тегла в множеството. Ако няколко елемента имат еднакво тегло, те се подреждат в азбучен ред възходящо; ZREM; ZCARD; ZSCORE - връща теглото на елемент; ZRANK/ZREVRANK - връща позицията на елемент в множество съответно при възходящо/низходящо сортиране; ZINCRBY - променя теглото на елемент, като към теглото му се прибавя положително или отрицателно реално число; ZCOUNT - връща броя на елементите с тегло между зададени стойности. По подразбиране интервалът е затворен, но може да се укаже и отворен интервал. Като граници на интервала може да се задават "-inf" и "+inf" за задаване на безкрайно малко и безкрайно голямо тегло.; ZLEXCOUNT - използва се само, ако теглата на всички елементи са еднакви. В този случай подредбата на елементите е във възходящ азбучен ред.; ZRANGE/ ZREVRANGE - извежда елементите от една позиция до друга позиция включително, които са сортирани възходящо/низходящо; ZRANGEBYLEX/ ZREVRANGEBYLEX; ZRANGEBYSCORE/ ZREVRANGEBYSCORE; ZREMRANGEBYRANK,

ZREMRANGEBYLEX, ZREMRANGEBYSCORE - премахват елементи от сортирано множество и връщат броя на премахнатите елементи; ZUNIONSTORE, ZINTERSTORE - команди за обединение и сечение на няколко множества със записване на резултата в друго множество.

От горепосочения кратък списък, в който са включени само по-интересните по наше усмотрение команди, можем да извадим заключението, че NoSQL системата Redis предлага не само много, но и разнообразни възможности. Различни по сложност абстрактни структури от данни са имплементирани и са предоставени за директно използване, при което не само се улеснява работата на разработчиците на системи, но и се преминава на качествено ново ниво при проектирането на сложни приложения за работа в реално време.

Въпреки интересните, от програмна гледна точка, възможности, които предлагат NoSQL системите, класическият подход за организация на големи обеми от данни е чрез **сървър на система за управление на релационни бази от данни** (СУРБД), т.е. работещо отделно сървърно приложение, което през мрежова връзка приема SQL заявки и връща резултат (Димитров, 2015). В този случай сървърът на СУРБД може физически да е разположен на друга машина и по този начин да се разпредели натоварването на сървърното приложение.

В последните две десетилетия СУРБД MySQL се ползва с голяма популярност сред разработчиците на уеб сървърни приложения[1]. Така например, отново според авторитетно изследване на австрийската консултантска фирма solidIT (DB-Engines, 2017b), към началото на 2018 г., най-популярната релационна система за управление на бази от данни с отворен код е MySQL (на първо място е системата Oracle, но тя не е с отворен код).

Сървърът на MySQL използва протокол TCP/IP за комуникация с

[1] Един от важните аспекти при администриране на система за управление на база от данни са процесите по архивиране и възстановяване - виж: (Куюмджиев, 2019).

клиента като приема заявки на определен порт. Съществена разлика с вече разгледаната SQLite е, че MySQL е многопотребителска система, което дава възможност за контрол до какви данни определен потребител ще има достъп и какви операции може да извършва с тях. По отношение на производителността - някой тестове показват по-добра производителност на SQLite, като за най-често използваните SQL команди тя е по-висока от тази на MySQL. За други заявки обаче производителността е по-ниска (Sqlite.org, 2017), т.е. еднозначно не могат да се направят никакви изводи за производителността, тъй като тя зависи от множество фактори. Тъй като релационните бази от данни широко се използват в практиката, тук няма да се спираме подробно на тях, въпреки че и те се развиват през годините, но с по-слаби темпове от NoSQL системите.

В заключение, изборът на конкретен подход при организацията на работа с данните при многослойна архитектура зависи от множество фактори. Използване на сървър на приложения Node.js дава възможност за прилагане на различни подходи чрез синхронно и асинхронно изпълнение на операциите, което води до по-добро използване на изчислителните ресурси, а това е особено важно при създаването на приложения за работа в реално време.

3.3. Кеширане на отговори на SQL заявки чрез използване на NoSQL система

При големи обеми данни скоростта при обработване на заявките понякога се явява от висока степен на важност за работа в реално време и минимизирането на времето за отговор може да се реализира посредством използването на структура от тип кеш. Би могло резултатите от бавно изпълняваните SQL заявки да се съхраняват в междинно хранилище с многократно по-малко време за отговор от тип NoSQL, в което да се намира определено подмножество от всички данни

в базата от данни.

Използването на кеша може да се извършва по следния начин: заявките, подобни на "SELECT name FROM users WHERE id = 111;" стават ключ в хеш-таблица, а резултатът от изпълнението на заявката - стойност, съответстваща на ключа. Ако ключът съществува, директно се използва кеширания резултат. Ако ключът не съществува, се изпраща SQL заявка и след това заявката и резултата се добавят в хеш-таблицата. Резултатът може да се запише като низ, което налага данните да се сериализират по подходящ начин, за да могат да се извличат отделни колони и редове.

При този подход е подходящо да се кешират резултантни таблици, съдържащи повече записи, вместо такива, съдържащи един или няколко записа, тъй като комбинираното натоварване при извличане на множество записи поединично е по-голямо от груповото им извличане на един път. Например заявката "SELECT name FROM users WHERE id = 111;" може да се преобразува в "SELECT name FROM users WHERE id >= 100 and id <= 200;" с последващо извличане на нужния запис по програмен път от кеша. По този начин значително се увеличава вероятността търсените данни да се намират в кеша.

Кеширането може да се извършва както в оперативната памет, така и на твърдия диск (Koltsidas&Viglas, 2011; Islam et al., 2015). Неудобството при кеширане в паметта е, че липсва постоянно съхраняване на данните в кеша, при което натоварването в първоначалния момент при запълване на кеша се увеличава[1]. Съхраняването на данните на кеш във файл спрямо паметта осигурява сравнително постоянна производителност през големи интервали от време и възможност за съхраняване на по-големи по обем данни.

[1] Този проблем, известен още като "студен кеш" представлява първоначално празен кеш, чието първоначално запълване отнема повече изчислителни ресурси от обичайното. За избягване на тази ситуация кеша предварително постепенно се "подгрява" с произволни заявки и след това се използва за нормална работа.

Недостатък е по-ниската скорост на работа, свързана с по-ниската скорост на обмен на данни от периферните устройства спрямо оперативната памет.

Трябва да се има предвид, че някои видове заявки не са подходящи за комбинирано използване с кеш. Това обикновено се случва когато данните по-често се обновяват и по-рядко се четат. Затова е необходимо предварително да се определи резултатите от кои заявки ще се кешират и на кои не. Според нас, това определяне може да стане в резултат на провеждането на тестове, които да покажат ефекта от използването на кеш при гранични условия - най-лош и най-добър случай, при което очакванията за разлика в производителността при реална система ще попадат в така определения диапазон. Освен това резултатите трябва да се трансформират от абсолютни стойности, които са във вид на "брой операции за единица време", в относителни.

Предлагаме този показател да се нарича **"относителна производителност"**, при което се съпоставят резултатите между две системи, получени при тестване при максимално близки хардуерни, софтуерни и други условия. В идеалния случай това означава изпълнение на тестовете по едно и също време, на един и същи хардуер, с общ език за програмиране и тип на извършвани операции. В зависимост от типа и сложността на операциите е възможно да се получат различни резултати, тъй като едни системи са по-добре оптимизирани да изпълняват определени дейности, докато други системи - други дейности. В резултат се получават стойности, които варират в широки граници, но в определен диапазон.

Според нас границите на този диапазон могат да бъдат определени, като това ще улесни определянето в кои случаи има смисъл да се прилага кеширане.

За да поясним казаното по-горе, ще определим долната граница на диапазона за относителна производителност при използване на

MySQL, съпоставено с Redis, които бяха разгледани по-напред. Разработили сме модул за тестване на бързодействието на MySQL и Redis на базата на Java, с цел приложението да може да се използва и на други платформи. Отчетени са особеностите при създаване на модули за тестване на Java (Boyer, 2008), като е направен опит да се минимизират проблемите, свързани със зареждането на клас-файловете, действието на Just-In-Time компилатора, автоматичното освобождаване на ресурси и прочие.

Таблицата в MySQL, използвана при тестовете е създадена по подобие на DBM базите от данни с две полета - целочислово - с първичен ключ, и текстово от тип varchar. По този начин структурата на таблицата се доближава максимално близко до NoSQL системите от тип ключ/стойност и по този начин може да се сравни "чистата производителност" при достъп до данните, без да се използва сложна релационна алгебра.

При четене на данните се използва опростен алгоритъм за генериране на псевдослучайни числа с цел заявките да са за записи, които не са последователно разположени, по подобие на реално работещите системи. Опити да се елиминира кеширането от страна на процесора, на входно-изходните операции от страна на операционната система или от страна на контролера на твърдия диск, не са правени.

Методите за добавяне на записи и за четене на записи в MySQL, които са най-съществената част от програмния код при тестове за определяне на относителна производителност са следните:

```
static long insertInMysql(int count) throws SQLException {
  Connection dbCon = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/benchmark",
    "benchmark",
    "123");
  PreparedStatement  stmt;
```

```
    stmt = dbCon.prepareStatement("TRUNCATE TABLE test");
    stmt.executeUpdate();

    long start = System.currentTimeMillis();
    for(int i=1; i<=count; i++) {
      stmt = dbCon.prepareStatement("INSERT INTO test VALUES (" + i + ", '" + i +
TXT + "')");
      stmt.executeUpdate();
      stmt.close();
    }
    long end = System.currentTimeMillis();

    dbCon.close();

    return end-start;
}

static long readFromMysql(int count) throws SQLException {
    Connection dbCon =
DriverManager.getConnection("jdbc:mysql://localhost:3306/benchmark", "benchmark",
"123");
    PreparedStatement  stmt;
    ResultSet rs = null;
    int id;

    long start = System.currentTimeMillis();
    for(int i=0; i<count; i++) {
      id = 1+i%200*i%200;
      stmt = dbCon.prepareStatement("SELECT data FROM test WHERE id = " + id + "
```

```
LIMIT 1");
      rs = stmt.executeQuery();
      rs.close();
      stmt.close();
    }
    long end = System.currentTimeMillis();

    dbCon.close();

    return end-start;
}
```

Методите за добавяне на записи и за четене на записи в Redis, които са най-съществената част от програменния код при тестове за определяне на относителна производитлност са следните:

```
static long insertInRedis(int count) {
    Jedis jedis = new Jedis("localhost");
    jedis.connect();

    jedis.flushDB();

    long start = System.currentTimeMillis();
    for(int i=0; i<count; i++) {
      jedis.set("" + i, i + TXT);
    }
    long end = System.currentTimeMillis();

    return end-start;
}
```

```
static long readFromRedis(int count) {
  Jedis jedis = new Jedis("localhost");
  jedis.connect();
  String v = "";
  int id;

  long start = System.currentTimeMillis();
  for(int i=0; i<count; i++) {
    id = 1+i%200*i%200;
    v = jedis.get(""+id);
  }
  long end = System.currentTimeMillis();

  return end-start;
}
```

Както се вижда, чрез предаване на аргументи на функциите се задава броя на записаните записи/стойности и броя на прочетените записи/стойности (при всички тестове сме ползвали 10 000 четения). Останалата част от кода не го публикуваме, тъй като не е съществен и представлява импортиране на пакети, декларация на константи и извиквания на горепосочените функции, след изчакване от 10 секунди между отделните етапи на работа, за да може операционната система евентуално да завърши някои операции.

Тестовете са проведени по следната схема: по дадени настройки за брой записи/стойности приложението се стартира три пъти с цел "подгряване" на оперативната памет и периферните устройства. Всеки тест се стартира 5 пъти, след което се прилага статистическия метод тримирано осредняване (още известно и като олимпийско осредняване) - най-малките и най-големите стойности се премахват и от останалите се намира средно аритметично. Броя записи/стойности нараства

експоненциално от 1 до 1 000 000. Резултатите са показани в две таблици. В таблица 3.1 са показани резултатите при запис, а в таблица 3.2 - резултатите при четене на 10 000 записа от MySQL или стойности от Redis. Флуктуаците в числата са в нормални граници при подобен род тестове.

Използваната при тестовете Java виртуална машина е Java SE Runtime Environment (build 1.7.0_40-b43), Java HotSpot Client VM (build 24.0-b56, mixed mode, sharing), използвания компилатор - JDK SE 1.7.0_40. СУРБД MySQL е версия MySQL Server 5.6.11 . Като драйвер за връзка с MySQL е използван "JDBC Driver for MySQL 5.1.26" . За NoSQL Redis сме използвали компилирана версия - Redis-2.4.5 с непроменяни настройки по подразбиране. За драйвер-клиент е използван Jedis 2.1.0 . Тестовете са проведени на ОС Windows 7 Professional, x86; процесор Intel i5-2430M@2.40GHz; RAM 4GB.

Таблица 3.1. Относителна производителност на MySQL и Redis при запис

Брой записи/ стойности	MySQL		Redis		Относителна производителност при запис MySQL:Redis
	Време за изпълнение	Брой записи в секунда	Време за изпълнение	Брой записи в секунда	
1	0 ms	-	0 ms	-	-
10	31 ms	323	0 ms	-	-
100	156 ms	641	0 ms	-	-
1 000	1,3 s	745	47 ms	21 277	1:29
10 000	13 s	763	0,4 s	24 631	1:32
100 000	140 s	713	4 s	24 468	1:34
1 000 000	26,2 min	636	39 s	25 589	1:40

Таблица 3.2. Относителна производителност на MySQL и Redis при четене на 10 000 записа/стойности

Брой записи/ стойности	MySQL		Redis		Относителна производителност при четене MySQL:Redis
	Време за изпълнение	Брой четения в секунда	Време за изпълнение	Брой четения в секунда	
1	1,2 s	8 224	0,4 s	22 936	1:3
10	1,2 s	8 326	0,4 s	23 753	1:3
100	1,2 s	8 326	0,4 s	22 883	1:3
1 000	1,2 s	8 439	0,4 s	22 883	1:3
10 000	1,2 s	8 117	0,4 s	22 883	1:3
100 000	1,2 s	8 479	0,4 s	23 753	1:3
1 000 000	1,2 s	8 217	0,4 s	22 935	1:3

Представените данни ни дават основание да направим следните изводи:

1. MySQL е десетки пъти по-бавен при запис на данни, спрямо Redis, като при увеличаване на броя записи в таблицата, тази тенденция се засилва. Долният диапазон за относителна производителност при запис MySQL:Redis е около 1:30 - 1:40, като при усложняване на структурата на добавяните записи (брой колони, тип данни, индексни полета и т.н.) този показател ще се влошава за MySQL, т.е. ще е над 1:40. Това се дължи както на сигурния начин, по който се изпълняват SQL-заявките, така и на богатите възможности на релационния модел.

2. Долният диапазон за относителна производителност при четене MySQL:Redis е около 1:3. И при двете системи се забелязва, че обемът на съхраняваните данни не оказва съществено влияние върху времето за търсене на нужния запис или ключ, и то е почти постоянно. При увеличаване на сложността на заявките чрез използване на пълните възможности на SQL, този показател отново ще се влошава за MySQL,

при което съотношението ще бъде над 1:3.

3. При положение, че записването на данни в Redis отнема незначително време спрямо подобна операция в MySQL, добавянето на кеширащ слой, използващ NoSQL, в едно приложение е обосновано при определена степен на вероятност данните да се намират в кеша и да не се налага изпращане на заявка до СУРБД. В този конкретен случай полза ще има при съотношение 1 запис, последван от поне 2,5 четения, т.е. при вероятност за намиране на данни в кеша над 60%[1]. Приложими са следните разчети на база емпиричните стойности от таблица 3.1 и таблица 3.2:

а) без кеширане в MySQL: 1 записване, последвано от "N" четения, повтарящо се 10000 пъти:

13 + 1,2xN

, т.е. при 1 четене (N=1) - времето за изпълнение ще е 14,2 s, при 2 четения (N=2) - 15,4 s, при 3 - 16,6 s, при 4 - 17,8 s, и т.н.

б) с кеширане чрез Redis, схемата е следната: 1 записване в MySQL, 1 изтриване в Redis. Следва алтернативно: 1 четене от Redis, 1 четене от MySQL, 1 записване в Redis, а при следващи четения - само 1 четене от Redis, като при "N"-четения първата алтернатива се изпълнява веднъж, а втората алтернатива се изпълнява N-1 пъти:

(13 + 0,4) + (0,4 + 1,2 + 0,4) + 0,4x(N-1)

, т.е. при 1 четене (N=1) - времето за изпълнение ще е 15,4 s, при 2 четения - 15,8 s, при 3 - 16,2 s, при 4 - 16,6 s, и т.н.

Вижда се, че при над 3 четения включително (точната стойност, както беше споменато по-горе, е 2,5 четения) сумарното време за изпълнение при наличие на кеш започва да намалява спрямо случая без кеш. Можем да заключим, че при тази ситуация, използването на кеш започва да носи ползи за увеличаване на производителността при обработка на данните.

[1] Получено е след приравняване на двата израза дадени по-долу и решаване на уравнението.

ГЛАВА ЧЕТВЪРТА. ИНСТРУМЕНТАЛНИ СРЕДСТВА ЗА РЕАЛИЗАЦИЯ НА СЪРВЪРА НА ПРИЛОЖЕНИЯ НА ИНФОРМАЦИОННАТА СИСТЕМА

4.1. Сървър на приложения от тип "event loop"

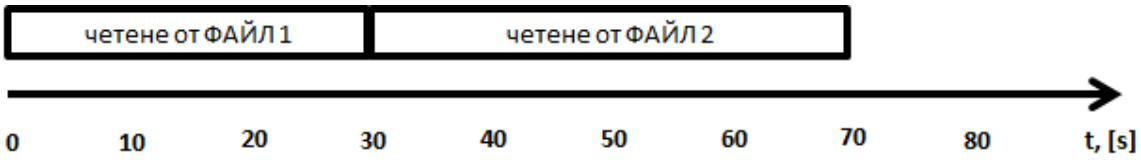
Както е известно, езикът за програмиране JavaScript е създаден от компанията Netscape през 1995 г. с цел създаване на програми от страна на уеб клиента. През 2008 г. Google предлага на потребителите своя браузър Chrome, в който производителността на JavaScript е съществено подобрена. Използваната виртуална машина V8 компилира програмния код в изпълним код и той се изпълнява с по-голяма скорост спрямо интерпретирания (Developers.google.com, 2018).

Платформата Node.js (Nodejs.org, 2018) се появява през 2009 и се базира на виртуалната машина V8, използвана в браузъра Google, като на практика се дава възможност за създаване на програми, написани на JavaScript, и от страна на уеб сървъра. Важен фактор за разпространението на Node.js е наличието и свободния достъп до множество програмни библиотеки - модули, чрез които се решават широк кръг от задачи. Към началото на 2018 г. има над 470 хил. модула, публикувани в NPM (Npmjs.com, 2018a), спрямо 230 хил. през 2016 г. (Wittern et al., 2016), и техният брой постоянно нараства, което от своя страна затруднява намирането на подходящия модул за конкретен проект. Основен проблем при използването на модули е, че част от тях дублират по функционалност други модули и това създава затруднения кой точно модул измежду няколко сходни да се избере (Kula et al., 2017).

Подобно на други езици за програмиране, в Node.js няма вградени функции за входно-изходни операции и затова се налага включване на програмни библиотеки, които са във външни файлове. За целта се използва глобалната вградена функция require(), която е подобна на ключовите думи #include в C/C++, import в Java или using в C#.

Основна особеност на Node.js е, че входно-изходните операции могат да се извършват по два начина - **синхронно** (блокиращо) и **асинхронно** (неблокиращо), и съответно се използват различни функции. Трябва да се има предвид, че разликата не е само в синтаксиса, но и в начина и последователността на изпълнение на операциите и това оказва голямо значение за производителността на сървърното приложение, което активно извършва такива операции (Tilkov&Vinoski, 2010; Loring et al., 2017).

Синхронните входно-изходни операции блокират изпълнението на програмата, т.е. тя спира изпълнението си, докато не завърши започната операция и чак когато тя приключи, тогава се преминава към изпълнението на следващата операция (виж Приложение 7). В периода от време от започване на операцията, примерно по четене на съдържанието на файла, до нейното приключване, програмата е блокирана и ако има нови събития, на които тя трябва да реагира, те се натрупват на опашката за събития, което означава отлагане за бъдещ момент на тяхното обработване, а именно докато входно-изходната операция не приключи. Например, ако трябва да са прочете съдържанието на два файла, изпълнението във времето би изглеждало по начина показан на фигура 4.1.

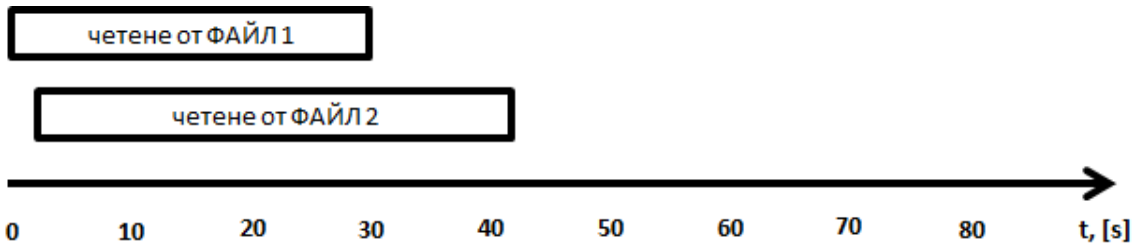


Фигура 4.1. Времедиаграма на синхронно (блокиращо) четене от два файла.

Възможно е операциите да се извършат асинхронно, при което след като започне изпълнението на едната операция, изпълнението на програмата да продължи и да започне изпълнението на втората

операция. Изпълнението в този случай във времето би изглеждало по начина показан на фигура 4.2.

При асинхронното изпълнение извършването на двете операции ще отнеме общо по-малко време.



Фигура 4.2. Асинхронно (неблокиращо) четене от два файла.

Прочитането на съдържанието на файл може да продължи неопределен период от време и това е характерно за всички входно-изходни операции. Причините за забавяне на изпълнението на входно-изходните операции могат да са най-различни, например операционната система може да извършва операции по дефрагментиране на дисковото пространство, при което операциите четене и запис значително се забавят. Възможно е да се извършват интензивни входно-изходни (суапови) операции във връзка с поддържането на виртуалната памет - прехвърляне на големи блокове от данни от оперативната памет на външното запаметяващо устройство и обратно. Възможно е в същия момент много приложения да се опитват също да извършват входно-изходни операции и т.н.

В други езици за програмиране обикновено изпълнението на "бавния" програмен код, извършващ входно-изходни операции се изнася в отделна нишка, т.е. използва се многонишково програмиране за успоредно изпълнение на няколко операции едновременно. В Node.js се използва друг подход - т.нар. събитийно-ориентирано програмиране (Philip, 1998). При него, изпълнението на една програма зависи от възникването на събития, които при сървърната част на едно

приложение са главно събития, свързани с мрежова връзка, работа с файлове или бази от данни, т.е. началото или края на входно-изходни операции, таймери и прочие. При клиентската част, събитията са основно предизвикани от действията на потребителя - мишка, клавиатура, данни от различни сензори и т.н. и са в основата при програмиране на работата с графичния потребителски интерфейс.

При **събитийно-ориентираното програмиране** в главната програма се указва какъв програмен код да се изпълни при възникване на определено събитие, подход, известен още и като "прихващане на събития". От своя страна програмният код, обработващ определено събитие не трябва да се изпълнява дълъг период от време, тъй като програмата няма да може да обработва други събития в същото време. Синтактично това се реализира като се извикват функции, в които като параметри се задават евентуално идентификатор на събитието, което ще се прихваща, и функция за обратно извикване (Developer.mozilla.org, 2018). Вместо да се създава отделна функция е възможно да се използва т.нар. анонимна (без име) функция (виж Приложение 8).

Много често програмите, написани на Node.js първоначално, при слабо натоварване, работят нормално, но е възможно да са налице потенциални проблеми, дължащи се на асинхронния характер на извършваните действия (Davis et al., 2018; Chang et al., 2019). Дори при сравнително елементарни операции, като например отваряне на файл, за аргумент трябва да се предава функция, която да се изпълни след отваряне на файла. Както е известно, отварянето на файл е операция, която се извършва чрез взаимодействие с операционната система, която от своя страна управлява файловата система. Необходимо е да се намери файла в структурата от директории и поддиректории, да се сравнят правата, които има приложението и правата, дадени на файла, и прочие служебни процедури. Възможно е файлът да е разположен на бавно или много натоварено периферно устройство, така че отварянето му да заеме

продължителен период от време. Тези обстоятелства налагат при създаването на асинхронни програми да се отделя повече внимание за определяне на това кои операции са синхронни и кои асинхронни с цел да се предотврати наличието на трудно отстраними грешки в последствие, които ще се появяват на вероятностен принцип.

Вместо директна работа с текстови (виж Приложение 9) или двоични файлове, при някои приложения при съхраняването на данни е подходящо да се използват бази от данни от тип "SQL програмна библиотека". Характерно за **SQL програмните библиотеки за работа с релационни бази от данни** е, че те не изискват инсталирането на отделно сървърно приложение. Те директно работят с файлове, намиращи се в локалната файлова система, като се използва езика SQL за работа с данните. Независимостта от сървърни приложения, лесното използване и добрата производителност са сред най-важните фактори при избора на този вид съхраняване на данни.

Съществуват различни SQL програмни библиотеки[1], които използват различен файлов формат, заемат различен обем памет и имат различно бързодействие. Както вече беше посочено в предходната точка, за тези цели може да се използва библиотеката SQLite[2] и в практиката тя често се използва като т.нар. "вградена база от данни"[3]. За да може SQLite да се използва в програми на Node.js, трябва да се използва програмна библиотека (модул), който не се разпространява стандартно заедно с Node.js.

Инсталирането на допълнителни модули се извършва в команден ред с програмата npm (Npmjs.org, 2019), която се разпространява заедно

[1] За различни езици за програмиране могат да се използват например библиотеките: Apache Derby DB - http://db.apache.org/derby/, H2 Database Engine - http://www.h2database.com/, Oracle Berkeley DB - https://www.oracle.com/database/technologies/related/berkeleydb.html, DuckDB - https://duckdb.org/, Firebird - https://firebirdsql.org/ и др.

[2] Документацията е публикувана на (Sqlite.org, 2019).

[3] Има се предвид, че SQLite не е самостоятелно сървърно приложение, а програмна библиотека с отворен изходен код, написана на C.

с Node.js и се инсталира заедно с него[1]. На база наши наблюдения, можем да твърдим, че намирането и инсталирането на подходящ модул от сайта npmjs.com понякога е трудна задача, тъй като много често съществуват множество модули с едно и също предназначение, но с различни имена, степен на развитие, поддръжка, качество и т.н. Някои модули се променят често и се тестват само под UNIX/Linux и това затруднява създаването на приложния, работещи под Windows.

Други модули са спрели развитието си и са несъвместими с по-новите модули. Затова когато се търси в хранилището на **Node Packaged Modules,** по наше мнение е необходимо да се обърне внимание преди всичко на информацията по следните критерии: номер на версия и кога е имало обновяване (показател за актуалност); колко пъти е свалян модулът за последния ден, седмица и месец (показател за популярност); от кои модули зависи и кои модули зависят от него (показател за обвързаност). Можем да кажем, че колкото показателите за актуалност, популярност и обвързаност на един модул са по-големи, толкова вероятността да е намерен подходящият модул в "джунглата от модули"[2] е по-голяма.

При инсталиране на един модул по подразбиране се инсталират и всички други модули, които той използва и от които той зависи[3]. Инсталирането на модулите локално в директорията на приложението е предпочитан подход в практиката, въпреки че поради дублиране на едни и същи модули на различни места се увеличава използваното дисково пространство. Причината е, че така разработваното приложение и

[1] При инсталирането на модули първо трябва да се смени текущата директория - тя трябва да стане същата като тази на приложението, което ще ги използва, тъй като допълнителните модули се инсталират в поддиректория node_modules на текущата директория на приложението и функцията require() ще ги търси там.

[2] Тъй като липсва централизиран подход за систематизация, проверка или одобрение на един или друг модул, то сравнението с джунгла, където на пръв поглед съществува дезорганизация, е напълно подходящо.

[3] Инсталираните модули могат да се използват в програмите като се включат с функцията require(). Ако модулът е инсталиран на нестандартно място, трябва да се укаже пълният път до него.

инсталираните за него модули не оказват влияние и не се влияят от модулите, използвани от други приложения.

Съществен проблем е, че някои модули се развиват твърде интензивно - прибавят се и се премахват функционалности в сравнително кратки периоди от време, и това затруднява синхронизацията с версиите (Zapata et al., 2018). В Node.js е приет **принципът на семантичните версии (**Semver.org, 2019). При него версиите на всеки модул се записват примерно по следния начин: "sqlite3":"3.1.13" - MAJOR.MINOR.PATCH, където смисълът на секциите е следния: MAJOR - големи промени в приложния програмен интерфейс (API); MINOR - има добавена нова функционалност, но при запазване на съществуващата функционалност (промяна в API, но има обратната съвместимост); PATCH - няма добавена нова функционалност, а са отстранени само грешки (няма промяна в API).

Когато едно приложение или модул в Node.js се разпространява, е прието да се придружава от специален описателен файл с име package.json (Npmjs.com, 2018b), в който се документира не само от кои модули зависи неговата работа, но и от кои техни версии. Например за приложение, използващо sqlite3 ver.3.1.13 съдържанието на файла може да е следното:

```
{
    "name": "Example App",
    "version": "0.0.1",
    "dependencies": {
        "sqlite3": "3.1.13"
    }
}
```

Възможно е в полето "dependencies" да се укаже и обхват от версии, ако приложението е тествано и е сигурно, че програмният код ще работи и с други версии. Например:

```
"dependencies" : {
        "sqlite3" : "3.1.13"
        , "sqlite3" : ">=3.1.1 <3.1.13"
        , " sqlite3" : "3.0.0 - 3.99.99"
        , " sqlite3" : "~3.1" , " sqlite3" : "~3.1.10"
        , " sqlite3" : "3.x" , " sqlite3" : "3.1.x"
}
```

Подобно на работата с файлове, съществуват два начина за изпълнение на SQL заявките: последователно (синхронно) и паралелно във времето (асинхронно). По подразбиране всички заявки се изпълняват паралелно, т.е. заявката се предава за изпълнение и изпълнението на програмата продължава. При този случай, в зависимост от продължителността на изпълнение на заявките, е възможно заявки, които се срещат по-назад в програмния код да приключат изпълнението си преди тези, които се срещат по-напред (виж Приложение 10).

При последователното изпълнение в даден момент от време се изпълнява само една заявка и чак когато изпълнението й приключи се преминава към изпълнението на следващата. С методите .serialize() и .parallelize() на обекта от тип sqlite се превключва между последователно и успоредно (паралелно във времето) изпълнение.

При използване на файлове или SQL програмни библиотеки може данните да не се съхраняват локално, а на друга машина - файлов сървър. В този случай операционната система симулира работа с локална файлова система и всъщност уеб сървърът или сървърът на приложения влизат в ролята на клиент и чрез мрежова връзка се свързват с отдалечен файлов сървър. Този начин на работа изисква голяма пропускателна способност и няколко клиента могат да изчерпят капацитета на мрежата. Типични имплементации са протоколът SMB на IBM/Microsoft, който позволява на базата протокола NetBIOS или върху TCP/IP (при версии на Windows след XP) споделяне на файлове в мрежа.

Проектът SAMBA предлага едноименен продукт, който позволява споделянето на файлове и принтери да се осъществява и между други операционни системи, а не само в средата на Windows (Samba.org, 2019).

Друг вариант за реализиране на файлов сървър е чрез използване на т.нар. мрежова файлова система, за чието ползване е необходима идентификация на клиента. Например системи, базирани на протокола Network File System (NFS) на IETF (Shepler et al., 2010); OpenAFS на Университета Карнеги-Мелън и IBM (Openafs.org, 2019); Apple Filing Protocol (AFP) на Apple (Developer.apple.com, 2019); Open Enterprise Server на (Micro Focus, 2018) (в миналото Novell) и др. В последните години широко приложение, във връзка с обработката на т.нар. "големи данни" ("Big Data"), намира системата Hadoop Distributed File System (Hadoop.apache.org, 2019), която е с лиценз свободен софтуер и се използва активно в редица големи компании като Facebook, Yahoo, IBM, eBay и др. Можем да заключим, че изборът на удачен вариант за реализация на файлов сървър е сложна задача и поставя много практико-приложни въпроси, които не е възможно да бъдат обхванати в настоящето изложение.

4.2. Организация на работата при обработка на данни във формат JSON

Данните, извличани чрез приложен програмен интерфейс (API) от сървъри, обикновено са във формат JSON (последното е съкратено от "JavaScript Object Notation"). Форматът е отворен, текстов формат за обмен на данни, стандартизиран от IETF (Crockford, 2006) и ECMA на (Ecma International, 2017). Базиран е на синтаксиса за представяне на прости и сложни типове данни в JavaScript и за пръв път е представен през 2001 г. Данните сравнително лесно могат да се четат и променят с помощта на обикновен текстов редактор и са платформено независими. Първоначалното му предназначение е да се използва за предаване на

данни към уеб приложения на JavaScript, но намира широко приложение и в други езици за програмиране[1]. Въпреки, че не е задължително да се познава синтаксиса на JavaScript, за да се работи с JSON, то е силно препоръчително да се познават в детайли основните типове данни в JavaScript с цел успешно да могат да се четат структурирани данни с произволна сложност.

Както е известно, в JavaScript основните типове данни са: прости (единични) данни (цели числа и числа с плаваща точка, низове, логически тип и празна стойност) и сложни данни (масиви, обекти, многомерни и вложени масиви и обекти). Понеже масиви и обекти (асоциативни масиви) могат да се вграждат и комбинират, могат да се реализират най-различни дървовидни структури от данни с произволна сложност (Bourhis et al., 2017).

В клиентските уеб приложения данните във формат JSON пристигнат във вид на символен низ. Необходимо е низът да бъде предварително обработен, преди данните да могат да се използват. Обработката на низ с данни, кодирани във формат JSON при използване на възможностите на езика за програмиране JavaScript може да се извърши по два начина - чрез глобалната функция eval() и чрез вградения обект JSON (Петров, 2015b).

Подходът с **eval()** се базира на факта (Richards et al., 2011), че тъй като JSON е подмножество на синтаксиса на JavaScript, то данните могат да бъдат обработени, т.е. разположени в паметта във вид, годен за директно използване, чрез използване на вградената глобална функция eval(), при което предаваните данни (по същество символен низ) се третират като програмен код на JavaScript. Функцията eval() приема като низ програмен код, изпълнява го и връща стойност, която се присвоява на променлива. В последствие, чрез тази променлива директно се използват всички данни, които вече са разположени в паметта в тяхното

[1] За поддръжка от други среди и програми виж: (Json.org, 2019).

т.нар. "вътрешно представяне".

Универсалният подход с използване на eval() е следният:

```
s = '["Likes","Followers","Retweets","Tweets"]'; //данни във формат JSON
s = '(' + s + ')'; //обграждане в обли скоби
x = eval( s ); //изпълняване на програмен код
alert(x[2]); // резултат: Retweets
```

Поставянето на низа с JSON данните в обли скоби е специално заради обектите. Ако например има eval('{}'), то при изпълнението на този програмен код фигурните скоби ще се разглеждат като начало и край на празен програмен блок, а не като празен обект, и ще се върне като стойност undefined. Но ако има eval('({})'), то тук се указва съдържанието в облите скоби да се разглежда като аритметично-логически израз (в който не може да има програмен код) и ще се върне празен обект, какъвто всъщност е смисълът на тази конструкция според JSON. Ще дадем още един пояснителен пример във връзка със спецификите на синтаксиса:

```
s= '{  \
      "name": "Ivan", \
      "Followers": 23 \
}';
user = eval( '(' + s + ')' );
alert(user.name); //Ivan
```

Както се вижда, съдържанието на променливата s не представлява изпълним програмен код, с програмни редове, които могат да се изпълнят и ще се получи грешка. Единствено и само заради това, че фигурните скоби се използват и при ограждане на програмен код, и при изброяване на съдържанието на обекти, се налага добавянето на обли скоби.

Най-често в практиката данните във формат JSON се получават от сървърната страна, а не се задават в програмния код, както при този

пример. В тази връзка трябва винаги да се има предвид, че функцията eval() изпълнява програмен код, което крие опасност при несигурен източник на данните да е възможно да се предаде и злонамерен програмен код (Yue&Wang, 2013; Santos et al., 2015). В такива потенциално опасни ситуации не трябва да се използва eval(), а задължително обектът JSON, разгледан малко по-нататък. Ако не е възможно да се използва обектът JSON, може да се извърши допълнителна проверка с регулярен израз (Crockford, 2006):

```
var my_JSON_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(
text.replace(/"(\\.|[^"\\])*"/g, ''))) &&
eval('(' + text + ')');
```

Например:

```
//злонамерени данни
s = '{}(function() {alert(123);}())';
eval('(' + s + ')'); //123 - злонамереният програмен код се изпълнява
data = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(
s.replace(/"(\\.|[^"\\])*"/g, ''))) &&
eval('(' + s + ')'); //злонамереният програмен код не се изпълнява, в data има false.
```

Вграденият обект JSON съдържа два метода за работа с данни във формат JSON (Ecma International, 2011) - метод, с който данни от паметта се преобразуват в JSON низ, и метод за обратната операция: .stringify() - преобразува в JSON формат и връща низ, и .parse() - обработва низ с данни във формат JSON и връща стойност, която обикновено е от сложен тип данни и по-рядко една-единствена стойност.

От страна на уеб сървъра за Node.js не е необходимо да се използват допълнителни модули за работа с формата JSON. В повечето случаи използването на вградения обект JSON е достатъчно.

Вграденият обект JSON съдържа два метода за работа с данни във формат JSON - един метод, с който данни от паметта се преобразуват в JSON низ, и един метод за обратната операция (Ecma

International, 2011):

JSON.stringify(ПРОМЕНЛИВА [, replacer [, space]]) - преобразува съдържанието на ПРОМЕНЛИВА в JSON формат и връща низ. Първият параметър е задължителен и обикновено е сложен тип данни (масив или обект), но може и да е прост тип данни - String, Boolean, Number или null.

Параметърът replacer е незадължителен и може да е или функция, или масив.

- Ако е функция, приема два параметъра - ключ и стойност, и трябва да връща нова стойност за ключа. Ако се върне същата стойност, тя остава непроменена. Ако се върне undefined, двойката ключ-стойност се изключва от резултата.

- Ако е масив, трябва да съдържа списък кои ключове (или индекси) и съответните им стойност трябва да бъдат включени.

Параметърът space също е незадължителен и задава как да се оформи низа, така че да е по-прегледен за четене от човек и може да е низ или число. Ако е число, задава колко интервала да се използват за отместването в началото на всеки ред. Ако е зададен низ, задава какво да се използва за отместването. Например:

```
<script>
var tweet = {
    id: "90071992547409921",
    "user": {
        "screen_name": "Petrov",
        "followers_count": "55"
    },
    place: {
        "country": "Bulgaria",
        "place_type": "city"
    }
```

```
};
document.writeln("<pre>");
document.writeln( JSON.stringify(tweet) );
document.writeln( JSON.stringify(tweet, null, 3) );
document.writeln(
    JSON.stringify(
        tweet,
        ["user", " screen_name", "place", "country"],
        3
    )
);
document.writeln(
    JSON.stringify(
        tweet,
        function(key, value) {
            a = {
                user: true,
                place_type: true
            };
            if(key === "id")
                return "XXXXX";
            if(a.hasOwnProperty(key))
                return undefined;
            else
                return value;
        },
        3
    )
);
</script>
```

В резултат се извежда:

{"id":"90071992547409921","user":{"screen_name":"Petrov","followers_count":55},"pla
ce":{"country":"Bulgaria","place_type":"city"}}

```
{ //JSON.stringify(tweet, null, 3)
  "id": "90071992547409921",
  "user": {
    "screen_name": "Petrov",
    "followers_count": 55
  },
  "place": {
    "country": "Bulgaria",
    "place_type": "city"
  }
}
{ //JSON.stringify(tweet, ["user", "screen_name", "place", "country"], 3)
  "user": {
    "screen_name": "Petrov"
  },
  "place": {
    "country": "Bulgaria"
  }
}
{ //JSON.stringify(tweet, function(key, value)...
  "id": "XXXXX",
  "place": {
    "country": "Bulgaria"
  }
}
```

JSON.parse(НИЗ [, reviver]) - обработва низ с данни във формат JSON и връща стойност. Стойността обикновено е сложен тип данни и

по-рядко една-единствена стойност. Параметърът reviver не е задължителен и е функция с два параметъра - ключ и стойност, и връща нова стойност. Използва се, за да се преобразува стойността в нова стойност. Ако се върне същата стойност, тя остава непроменена. Ако се върне undefined, двойката ключ-стойност се изключва от резултата. Именно тази функция се препоръчва от добрите практики за използване вместо eval(). Например:

```
<script>
tweet = JSON.parse('{"id": "90071992547409921","user": {"screen_name":
"Petrov","followers_count": 55},"place": {"country": "Bulgaria","place_type": "city"}}');
document.writeln(tweet.user["followers_count"] + " #" + tweet.user.screen_name);
</script>
```

В резултат се извежда:

```
55 #Petrov
```

В практиката често се налага да се работи с дати. В JavaScript има вграден обект Date, но когато датата трябва да се предаде, тя обикновено се предава или като масив от числа, или като едно число, или като низ.

Когато се използва масив от числа, лесно може да се създаде нов обект на база на вградения обект Date чрез единия от конструкторите:

```
new Date(ГГГГ, М, Д, ч, м, с, мс)
```

Когато се използва едно число, то трябва да е броя милисекунди от 01.01.1970 г. (това е т.нар. система "Unix time") и може да се използва друг конструктор:

```
new Date(мс)
```

Когато се използва низ, най-добре е да се използва стандартен формат за представяне на датата, какъвто е ISO UTC, за да може низа също да се предаде на друг конструктор, подобно на предходните два случая. Функциите на Date .toISOString() и .toJSON() връщат дата като низ във формат ISO UTC, а конструктор на Date преобразува този низ във вътрешно представяне. Например:

```
<script>
s = '{ \
        "source":"Twitter for Mac", \
        "text":"Hello", \
        "date":"2018-10-20T08:50:40.000Z" \
        "created_at":"Tue Jan 01 22:44:03 +0000 2019" \
}';
d = JSON.parse(
        s,
        function(k, v) {
                if (k == 'date')
                        return new Date(v);
                if (k == 'created_at')
                        return new Date(Date.parse(v.replace(/( \+)/, ' UTC$1')));
                /* else */
                return v;
        }
);
alert( d.date.getFullYear() ); //да не се използва .getYear() !
</script>
```

Извежда се:

```
2018
```

4.3. Производителност на вариантите за обработка на данни във формат JSON

За разлика от строго типизираните езици за програмиране като C, C++, Java, C# и др. в JavaScript, както при повечето интерпретатори езици има по-голямо разнообразие от вградени (основни) типове данни. В JavaScript основните типове данни във връзка с моделиране чрез JSON са (Pandey&Pandey, 2017; Lv et al., 2018):

Прости (единични) данни:

- число - цели числа и числа с плаваща точка, например 123 или 3.1415926. Използваната разрядност при вътрешното представяне на числата е еднаква и за целите, и за реалните числа, и е 64 бита (1 бит за знака, 52 бита за мантисата, 11 бита за порядъка). Диапазоните са +/-9007199254740992 и +/-(от 5e-324 до 1.7976931348623157e+308). Много големи цели числа обикновено се използват като сесийни и други идентификатори. Десетичният разделител винаги е точка. Много големи числа могат да се представят и като символни низове. При работа с числа обикновено се използват вградените обекти Number и Math, както и някои глобални функции, като isFinite(), isNaN(), Number(), parseFloat(), parseInt(). Използват се и глобалните константи като Infinity, NaN и undefined за проверка съответно дали е възникнало препълване, дали стойност е валидно число и дали променлива има зададена стойност.

- низове - последователност от символи, оградени с двойни кавички или апострофи (няма разлика между тях), например "Това е символен низ". При JSON задължително се използват двойни кавички. При работа с низове обикновено се използват вградените обекти String и RegExp, както и някой глобални функции като decodeURI(), encodeURI(), eval() и др.

- логически - работи се с двете стойности true и false. Специално за работа с логически стойности съществува глобален обект Boolean, който не предлага големи възможности. Много често вместо този тип данни се използват числата 0 и 1 за представяне на логически стойности.

- празна стойност - null. Задава липса на стойност.

Сложни данни (съставни, включват в себе си други типове данни) - масиви, обекти (асоциативни масиви), многомерни и вложени масиви и обекти:

- масиви - характерно за JavaScript е, че масивите са динамични

масиви, т.е. броят на елементите може да се променя по време на изпълнение на програмата. В други езици за програмиране за тази цел се използват обекти от класове Vector, ArrayList или списък. Друга особеност е, че в един масив могат да се съхраняват данни от различен тип. Има различни начини за създаване на масиви. Единият е чрез използване на конструктора на вградения обект Array(). Например:

```
<script>
a = new Array();
//a = new Array(7); //задаване на първоначален брой елементи
a[0] = "Likes";
a[1] = 123;
a[2] = "Followers";
a[3] = 4567;
//a = new Array("Likes", 123, "Followers", 4567);
//a = [ "Likes", 123, "Followers", 4567 ];
for (i=0; i<a.length; i++)
	document.write(a[i] + "<br />");
</script>
```

Достъпът до елементите на масива става посредством индекс - число, започващо от 0.

- обекти - това са асоциативни масиви и в други езици за програмиране се наричат хешове, записи, структури, речници, представят се с класове Map, но в документацията на JavaScript се наименуват като обекти. Асоциативните масиви дават възможност вместо индекс - число за достъп до елементите да се използва ключ - уникален низ. В този случай елементите не са подредени и трябва да се използва друг подход за обхождане на всички елементи. Използва се модифициран вариант на оператора for - for (КЛЮЧ in МАСИВ). Например:

```
<script>
```

```
a = new Array();
a["key1"] = "Likes";
a["key2"] = 123;
a["key3"] = 45.67;
//a = { "key1": "Likes", "key2": 123, "key3": 45.67 };
for (k in a)
	document.write( a[k] + "<br />" );
</script>
```

Както се вижда, за създаването и на масиви, и на обекти, се използва вградения обект Array и определянето на точния вид на масива (дали е обикновен масив или обект) е в зависимост от контекста на операциите, които се извършват с него.

- многомерни масиви - масиви, които вместо единични стойности съдържат масиви (или обекти). В този случай многомерният масив представлява масив от масиви и в общия случай не може визуално да се представи примерно като таблица, тъй като броят на колоните във всеки ред може да варира. Например:

```
<script>
//обикновен двумерен масив
a = new Array();
a[0] = new Array( "Likes", 123, 45.67 );
a[1] = new Array( "Followers", 123, 45.67, "Text", 123 );
a[2] = new Array( "Retweets", 123, 45.67 );
document.write( a[1][4] + "<br />");
//обект (асоциативен масив) от масиви
b = new Array();
b['key1'] = new Array( "Likes", 123, 45.67 );
b['key2'] = new Array( "Followers", 123, 45.67, "Followers", 123 );
b['key3'] = new Array( "Retweets", 123, 45.67 );
document.write( b["key3"][2] + "<br />");
```

```
</script>
```

Понеже масиви и обекти (асоциативни масиви) могат да се вграждат и комбинират, могат да се реализират дървовидни структури от данни с произволна сложност.

```
<script>
a = ["Likes","Followers","Retweets","Tweets"]; // a е обикновен масив
alert(a[2]); //ccc
a["Likes"] = 111; // a става обект (асоциативен масив)
a["Followers"] = 222;
a["Retweets"] = 333;
alert(a["Followers"]); //222
alert(a.Likes); //111 - синтаксис еквивалентен на a["Likes"]
alert(a[2]); //Retweets - обекта запазва данните от масива
b = {
"Likes":111,
"Followers":222,
"Retweets":333
};
alert(b["Likes"]); //111
</script>
```

При създаване на приложения, работещи в реално време е възможно "тясно място" при обработката на данни да се окаже преобразуването на низ с данни, форматирани във формат JSON в обект, съдържащ структурата от данни, описана в низа. Поради тази причина е разработен тест, който цели да установи, кой подход на работа е най-бърз, и следователно най-подходящ за използване, при създаване на приложения за работа в реално време, при които има интензивна обработка на JSON низове. Резултатите от тестовете са показани на

таблица 4.1[1].

Тестовете са проведени със следните видове уеб браузъри: GC - Google Chrome Version 71.0.3578.98 (Official Build) (64-bit); ME - Microsoft Edge 42.17134.1.0 with Microsoft EdgeHTML 17.17134; IE - Internet Explorer 11.523.17134.0; MFF - Mozilla Firefox 64.0.2 (64-bits) ; Програмна среда NODE.JS - node-v10.15.0-win-x64.

Таблица 4.1. Операции в секунда при използване на различни подходи за преобразуване на JSON низ в програмен обект. Система: Intel Core i5-2400 CPU@3.10GHz, RAM 8GB. Избягване на кеширането: data.push(json_str.replace("2019", i));

браузър/среда / подход	GC	MFF	ME	IE	NODE
JSON.parse(str)	65359	50505	42735	51813	72993
	81301	**70922**	**49505**	44643	**86957**
	69930	72464	50251	**47619**	88496
	84034	72993	51546	40816	74074
	84746	17575	49261	54645	90090
(new Function("return " + str))()	9225	**1098**	4568	2919	9302
	9116	1067	3357	2830	9074
	117647	1061	3352	**2224**	**37594**
	119048	1129	**3401**	1911	90909
	119048	1124	3409	1673	112360
eval("(" + str + ")");	10753	45455	4963	4983	10695
	10309	**46948**	4596	**4980**	10384
	47847	47393	4560	4766	**100000**
	48309	38911	**4583**	4335	113636
	46729	47847	4570	5053	128205

[1] При създаване на тестове е възможно да се използва и експертен метод за оценка на софтуерната производителност - виж: (Сълов, 2014).

Всеки от серията тестове е проведен на два компютъра с различни параметри, но с еднаква операционна система - ОС Windows 10 Pro, 64 бита. Първият компютър е със следните характеристики: процесор Intel Core i5-2400 @3.10GHz, RAM 8GB и данните, получени чрез него са представени в таблица 4.1, таблица 4.2, таблица 4.3 и таблица 4.4. Вторият компютър е следните характеристики: процесор Intel i5-2430M@2.40GHz; RAM 4GB и данните, получени чрез него са представени в таблица 4.5, таблица 4.6, таблица 4.7 и таблица 4.8.

Всеки от тестовете се повтаря пет пъти, след което се избира медианата като средна стойност от проведения тест. Медианите са маркирани с удебелен шрифт и всички първичните данни от тестовете са представени в таблица 4.1, таблица 4.3, таблица 4.5 и таблица 4.7.

Въз основа на първичните данни чрез средно аритметично се определя средния брой на операциите в секунда за всеки един от трите подхода, след което се изчислява относителната скорост на операциите, спрямо най-бавния подход, представен като 100%. Тези данни са представени в таблица 4.2, таблица 4.4, таблица 4.6 и таблица 4.8.

Таблица 4.2. Относителна скорост на операциите от таблица 4.1.

подход	операции в секунда /средно аритм./	относителна скорост
JSON.parse(str)	**67261**	**208%**
(new Function("return " + str))()	32393	100%
eval("(" + str + ")");	40648	125%

Тестовете са проведени в два варианта: първи вариант, с опит за избягване на кеширането, като се обработват различаващи се JSON низове. За целта в масив се поставят променени JSON низове чрез програмния ред "data.push(json_str.replace("2019", i));". При вторият вариант се обработват последователно многократно едни и същи JSON

низове, но отново разположени в масив. Основната цел при този начин на провеждане на тестовете е да се установи дали и как точно влияе кеширането на резултатите от обработка на JSON низове при отделните видове браузъри.

Таблица 4.3. Операции в секунда при използване на различни подходи за преобразуване на JSON низ в програмен обект. Система: Intel Core i5-2400 CPU@3.10GHz, RAM 8GB. Без избягване на кеширането: data.push(json_str);

браузър/среда / подход	GC	MFF	ME	IE	NODE
JSON.parse(str)	84034	70423	47847	52910	90909
	85470	70922	51020	44643	91743
	86957	**72464**	**51282**	**51282**	**91743**
	84746	74074	54348	51282	81301
	85470	20619	54945	51020	91743
(new Function("return " + str))()	128205	1111	144928	108696	120482
	142857	**614**	181818	113636	140845
	161290	485	178571	108696	**147059**
	161290	560	**169492**	**113636**	153846
	166667	1094	166667	113636	172414
eval("(" + str + ")");	50761	46729	192308	82645	163934
	52356	49020	**208333**	123457	192308
	52083	48780	217391	149254	133333
	52356	44444	217391	**135135**	**181818**
	52632	**47619**	208333	140845	188679

Таблица 4.4. Относителна скорост на операциите от таблица 4.3.

подход	операции в секунда /средно аритм./	относителна скорост
JSON.parse(str)	70448	100%

(new Function("return " + str))()	118418	168%
eval("(" + str + ")");	**125052**	**178%**

Таблица 4.5. Операции в секунда при използване на различни подходи за преобразуване на JSON низ в програмен обект. Система: Intel i5-2430M@2.40GHz; RAM 4GB. Избягване на кеширането: data.push(json_str.replace("2019", i));

браузър/среда / подход	GC	MFF	ME	IE	NODE
JSON.parse(str)	44843	40486	33445	41322	59524
	66225	60976	40816	42373	71942
	67568	50505	**39370**	**34483**	**74074**
	69930	**57143**	38610	33333	79365
	70423	60606	39370	30211	78740
(new Function("return " + str))()	7391	**5952**	2861	2311	8091
	7278	6435	2616	2217	7868
	48309	4995	**2608**	**1753**	**32680**
	91743	4876	2575	1509	95238
	91743	6135	2568	1217	92593
eval("(" + str + ")");	8673	18215	3525	4027	8897
	8741	40650	3579	4095	8651
	40161	**38610**	**3574**	3798	**84746**
	39683	40161	3346	**4067**	109890
	40161	13106	3617	5784	109890

Таблица 4.6. Относителна скорост на операциите от таблица 4.5.

подход	операции в секунда /средно аритм./	относителна скорост
JSON.parse(str)	**54528**	**299%**
(new Function("return " + str))()	18260	100%
eval("(" + str + ")");	34136	187%

Таблица 4.7. Операции в секунда при използване на различни подходи за преобразуване на JSON низ в програмен обект. Система: Intel i5-2430M@2.40GHz; RAM 4GB. Без избягване на кеширането: data.push(json_str);

браузър/среда / подход	GC	MFF	ME	IE	NODE
JSON.parse(str)	64935	58824	36101	44643	76923
	69930	**59880**	46296	33670	77519
	70423	60241	45662	44053	**77519**
	72464	64103	42553	**42373**	81301
	72464	16207	**45045**	41322	78125
(new Function("return " + str))()	101010	829	125000	92593	101010
	112360	443	**138889**	96154	123457
	135135	317	142857	**78740**	**144928**
	140845	**580**	138889	98039	144928
	138889	823	140845	78740	144928
eval("(" + str + ")");	40984	39526	142857	64935	120482
	42017	42017	**156250**	93458	144928
	44053	41667	166667	111111	163934
	40984	37175	163934	**97087**	**161290**
	43668	**39683**	142857	104167	161290

Таблица 4.8. Относителна скорост на операциите от таблица 4.7.

подход	операции в секунда /средно аритм./	относителна скорост
JSON.parse(str)	59048	100%
(new Function("return " + str))()	**99654**	**169%**
eval("(" + str + ")");	**99265**	**168%**

Понеже операциите по обработка на символни низове са сравнително ресурсоемки, то е възможно производителите на браузъри и

на JavaScript интерпретатори да кешират или съхраняват в буфер вече обработен низ. В таблица 4.1, таблица 4.2, таблица 4.5 и таблица 4.6 са представени данни при избягване на кеширането чрез подаване за обработка на различни JSON низове, а в таблица 4.3, таблица 4.4, таблица 4.7 и таблица 4.8 - без избягване на кеширането и обработване на един и същ низ.

Трябва да се има предвид, че при варианта без избягване на кеширането, подаваните за обработване JSON низове са едни и същи като стойност, но се разполагат на различни адреси от паметта. Тази ситуация значително се различава от ситуацията, при която за многократна обработка се подава един и същ адрес от паметта (примерно една и съща скаларна променлива или текстова константа), при което сравнително лесно към този адрес може да се асоциира готов резултат от обработката, така че реално обработка да има само първия път.

Оригиналните данни в JSON формат, използвани при тестовете са представени в Приложение 11. Това е туит, представляващ годишна прогноза за цената на природния газ, дадена от президента на САЩ. Програмата, използвана при тестовете за производителност при обработка на данни във формат JSON при използване на различни програмни средства е представена в Приложение 12.

По отношение на кеширането на обработените низове, както споменахме, понеже операциите по обработка на символни низове са сравнително ресурсоемки, то производителите на браузъри кешират или съхраняват в буфер вече обработен низ. Това лесно се забелязва когато се съпоставят данните от таблица 4.2 и таблица 4.4, съответно и таблица 4.6 и таблица 4.8. С изключение на подхода с "JSON.parse(str)" (слабо нарастване) като цяло броя на операциите в секунда нарастват в пъти - от 3 до 5 пъти.

Според нас това означава, че е необходимо повишено внимание,

когато се провеждат подобни тестове, тъй като е възможно да се получат големи разлики в резултатите при провеждане на тестове и при работа в реална среда. Необясними резултати показва браузърът Mozilla Firefox при подход "(new Function("return " + str))()". В тестовете, проведени на различни компютри, при обработка на един и същ JSON низ резултатите са значително по-ниски (до 10 пъти разлика), отколкото при обработката на различни JSON низове. Според нас ще е изключително интересно да се установи причината за тази аномалия, но това не е в обхвата на настоящото изследване.

Резултатите от тестовете с избягване на кеширането, представени в таблица 4.1, таблица 4.2, таблица 4.5 и таблица 4.6, ни дават основание да направим следните изводи:

1. В случай, че има възможност за избор на браузър, то от таблица 4.1 и таблица 4.3 следва, че най-добри резултати се получават при Google Chrome и използване на подходи "JSON.parse(str)" и "(new Function("return " + str))()". При Google Chrome подходът "eval("(" + str + ")");" трябва да се избягва, тъй като той дава значително по-слаби резултати. Това обаче не се отнася за Node.js, въпреки че би трябвало да се очакват сходни резултати, тъй като Node.js е базиран на интерпретатора "V8", разработен от Google.

2. Браузърите Microsoft Edge и Internet Explorer при подходи "(new Function("return " + str))()" и " eval("(" + str + ")");" се представят от 10 до 50 пъти по-слабо, спрямо Google Chrome или спрямо използване на "JSON.parse(str)". При тях най-подходящ за използване е "JSON.parse(str)", а останалите подходи трябва да се избягват.

3. Браузърът Mozilla Firefox при подход "(new Function("return " + str))()" се представя от 10 до 50 пъти по-слабо, спрямо Google Chrome или спрямо използване на останалите два подхода. При него най-подходящ за използване е "JSON.parse(str)", а подход "(new Function("return " + str))()" трябва да се избягва.

4. В случай, че предварително не е известно какъв точно браузър ще се използва, най-добри резултати според данните в таблица 4.2 и таблица 4.6 биха се получили при използване на подход "JSON.parse(str)". Скоростта на обработка на JSON низове при него е средно около 2-3 пъти по-висока спрямо останалите подходи.

ГЛАВА ПЕТА. ПОДХОД ЗА ОБМЕН НА ДАННИ В РЕАЛНО ВРЕМЕ

5.1. Възможности за работа в реално време

Един от основните моменти в развитието на уеб технологиите е усъвършенстване на техниките, използвани при изграждане на уеб приложения, работещи в реално време. Подобни приложения се различават съществено от класическите разбирания за уеб сайт, доближавайки се като функционалност и поведение до т.нар. "десктоп" приложения. Една от задачите ни е да се идентифицират съвременните подходи за изграждане на уеб приложения в реално време в контекста на изграждането на информационни системи за предоставяне на финансови услуги в реално време и да се даде оценка на перспективността на някои нововъзникнали подходи.

Основно място за технологични подобрения се явяват уеб клиентът, уеб сървърът и протоколът, използван при комуникацията. Както вече беше споменато в четвърта глава, уеб клиентът представлява приложение (браузър или програма, изпълнявана от браузър), което започва мрежова комуникационна връзка с уеб сървър. Уеб клиентът обикновено получава от потребител заявки чрез графичен потребителски интерфейс. Потребителските заявки се преобразуват в заявки по определен уеб протокол, който се поддържа и от уеб сървъра. Такива уеб протоколи към настоящия момент са HTTP 0.9/1.0/1.1, HTTPS, HTTP/2, WebSocket, SSE и Push API. Активно се разработва и навлиза HTTP/3 (известен още и като "HTTP over QUIC", "HQ" или "H3"), а вероятно в бъдеще ще се появят и други протоколи.

Технологичните подобрения и при уеб клиентите, и при уеб сървърите, се състоят преди всичко в разширяване на поддръжката на нови уеб протоколи. Уеб протоколът, както вече беше споменато в четвърта глава, е система от правила, която определя как уеб клиентът и уеб сървърът да комуникират помежду си, в това число и

последователността на заявките и отговорите, формат на обменяните данни, времеви ограничения и прочие. Те се стандартизират от международната организация Internet Engineering Task Force (IETF, 2019) и са известни под името RFC (Request for Comments) (IETF, 2018).

За осъществяване на комуникацията между уеб клиента и уеб сървъра се използва мрежова връзка - сокет (Maata et al., 2017). От програмна гледна точка последната позволява чрез използване на приложен програмен интерфейс (API), предлаган от операционната система, да се осъществи комуникация между две различни приложения, без значение къде се намират физически и логически в компютърната мрежа (Wang, 2018). Тъй като при клиент-сървър програмирането се използват две различни приложения за изпълнението на едно общо действие, много често тези приложения са териториално отдалечени на големи разстояния, при което съвместната работа се разпределя във времето и се налага изчакване между отделните етапи на съвместната работа.

Основен проблем при използването на клиент-сървър приложения е, че големият брой едновременно изпратени заявки за мрежови връзки от страна на множество клиенти към един единствен сървър могат да създадат голямо натоварване за машината на сървъра и при изчерпване на сървърните ресурси може да се стигне до ситуация, при която клиентски заявки не се обслужват в приемлив период от време или веднага се отхвърлят. Нещо повече - понякога се стига до ситуация, при която необслужените заявки се изпращат непрекъснато, без изчакване, отново и отново от клиентите към сървъра, което още повече натоварва сървъра и той започва да отхвърля все повече заявки за единица време. Цикълът се повтаря деградиращо и тази ситуация в практиката е известна като "отказ от обслужване" (Denial of Service, DOS) - сървърната машина работи интензивно, но голяма част от клиентските заявки остават необслужени (Adi et al., 2017;

Tripathi&Hubballi, 2018).

Затова в последните години в практиката все повече се използват т.нар. **"облачни услуги"**, при които използваните от приложенията системни ресурси, оказващи съществено влияние върху бързодействието - процесорно време и памет, могат да се променят динамично за определени приложения (Zekri et al., 2017; Bhushan&Gupta, 2019) и така да се осигури мащабируемост[1]. Това се постига като се използват различни техники: модулно нарастване на достъпната за дадено приложение памет и отделено процесорно време; автоматично прехвърляне на приложението на друга машина в "облака", която е с по-добри характеристики (повече памет, повече процесори или ядра, по-висока производителност) или чрез разпределяне на натоварването върху няколко машини едновременно в същия или в различни географски райони на "облака".

При работа в режим реално време е необходимо да се избере стратегия, по която резултатите от обработката на данни от сървърната страна на разпределеното приложение ще достигнат до своя потребител. Необходимо е да се осигури както непрекъснатост на информационния поток между сървъра и клиента, така и да се отчита изискването, че интервалът от време за реакция (времезабавянето) трябва да е колкото се може по-малък. За целта, при създаването на информационната система, могат да се използват няколко различни стратегии, които условно групираме в три основни категории: **класически стратегии, съвременни стратегии и стратегии в перспектива**.

Разликата между отделните категории технологични стратегии е, че първата има универсален характер и може да се прилага за широк кръг от устройства и браузъри, но ефективността по отношение на

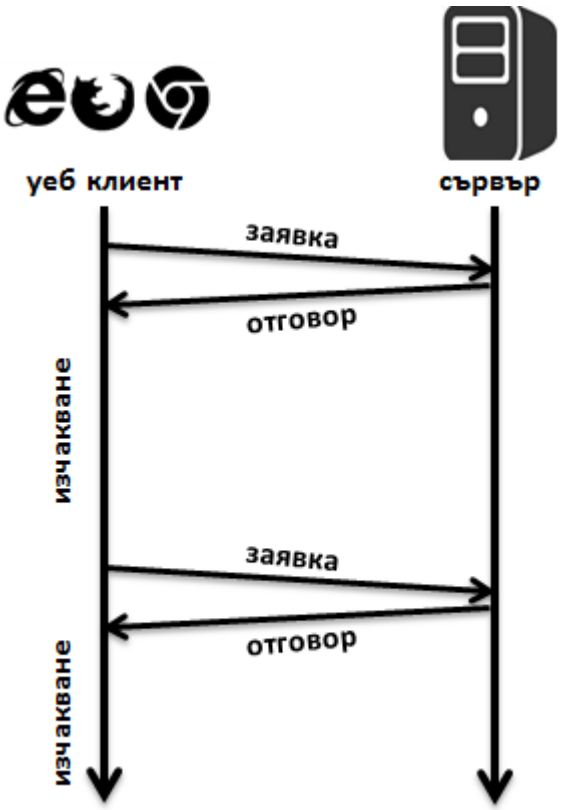
[1] Под мащабируемост разбираме възможността сървърната част да може да поема динамични увеличения на заявките от страна на клиентите и обемите обработвани данни, при запазване на производителността на цялата клиент-сървър система на високо ниво и без забележимо да се нарушава или прекъсва работата с клиентите.

изчислителни ресурси и обем трафик не е добра. При втората категория стратегии се използват съвременни и ефективни технологии, но все още има в употреба устройства и браузъри, които не ги поддържат - това са т.нар "морално остарели" устройства. Третата категория стратегии все още е технологично неприложима за голяма част от употребяваните устройства и браузъри, но очакваме това да се промени в бъдеще.

5.2. Класически стратегии "Издърпване" и "Дълго запитване"

Исторически най-старият подход за постигането на целта - работа в реално време, е зареждането в уеб браузъра на страница, която се обновява постоянно, през определен интервал от време (Bozdag et al., 2007; Krawiec et al., 2017). Това е т.нар. **техника "издърпване" - "pull"** (фигура 5.1), която се реализира чрез HTML тага meta (например: <meta http-equiv="refresh" content="3" />) или чрез използване на таймер в JavaScript (например: setInterval(function() { window.location.reload(true); }, 3000);).

При първия вариант не се налага програмиране от страна на клиента, а във втория се използва език за програмиране, като в годините до 2005 г. освен JavaScript се е използвал и VBScript, главно за Internet Explorer. При този подход цялата или основната част от логиката е изнесена на страната на сървъра и натоварването му нараства значително при намаляване на времето за обновяване на уеб страницата и при нарастване на броя на уеб клиентите (Martin-Flatin, 1999). Допълнителен проблем е, че при неуспешно зареждане на поредната страница, се налага потребителят собственоръчно да презареди страницата, тъй като изведеното съобщение за грешка прекъсва цикъла на автоматичното презареждане.



Фигура 5.1. Времедиаграма на комуникацията между уеб клиента и уеб сървъра при техниката "издърпване" - "pull".

Определянето на интервала от време за презареждане се осъществява чрез съобразяване с две взаимно противоречащи си условия: висока степен на актуалност на данните (минимална латентност) при стремеж за малко натоварване на уеб сървъра. По-дългият интервал може да доведе до пропускане на точния момент на възникване на някое събитие при сървъра, а по-късият интервал води до по-голям трафик и по-голямо натоварване на уеб сървъра. В идеалния случай интервалът на презареждане трябва да съвпада с интервала на възникване на събития при сървъра.

Еволюция на гореописания подход е използването на HTML фреймове - чрез тагове frameset или iframe, като в родителския фрейм се поставя програмния код за обновяване на видим или невидим подчинен фрейм чрез таймер (Yao, Sun, Hu, Zhu, Ni, 2008). Тъй като родителският

фрейм се зарежда еднократно в началото и в последствие таймерът работи постоянно, се избягва проблемът при неуспешно зареждане на поредна страница във фрейма. При такава ситуация, дори и в подчинения фрейм да възникне грешка, следващото задействане на таймера в родителския фрейм ще направи опит за ново зареждане и цикъла на презареждане няма да се прекъсне. Тук отново цялата или по-голяма част от логиката е от страна на сървъра, т.е. това е т.нар. архитектура "тънък клиент" (виж Приложение 1).

Следващ етап от еволюцията на уеб приложенията, работещи в реално време, е намаляване или премахване на изпращането на едни и същи данни многократно. Обикновено при уеб приложения в реално време структурата на уеб страницата се запазва в течение на времето, а се променят само показваните данни. Ако шаблонът на страницата, указващ положението и оформлението на отделните данни, се изпрати еднократно, а в отговор на всяка заявка се изпращат само променящите се данни, то трафикът между сървъра и клиента може да спадне значително, което влияе положително и върху намаляване на времевия интервал от изпращане на HTTP заявката до получаване на отговора. Допълнителен положителен ефект има при уеб сървъра, тъй като отпада операцията вмъкване на данни между HTML тагове, което на практика представлява елементарно слепване на символни низове или поредица от операции търсене и заместване при използване на шаблони.

Характерно и при двата варианта е, че се използва значително количество оперативна памет и честото й динамично заемане и освобождаване намалява производителността на сървъра като цяло. В тежки случаи може да се стигне и до използване на тази част от виртуалната памет, която е разположена на периферно устройство, което още по-рязко намалява производителността. Изпращането само на данните, и то само на тези, които са се променили, прехвърля значителна част от обработката на данните от сървъра към клиента и се отива към

т.нар. архитектура "дебел клиент". В идеалния случай от страна на уеб сървъра само се извличат данните (от файлове, чрез заявка до сървър на база от данни или чрез NoSQL) и веднага се връща отговор на клиента без някаква допълнителна обработка.

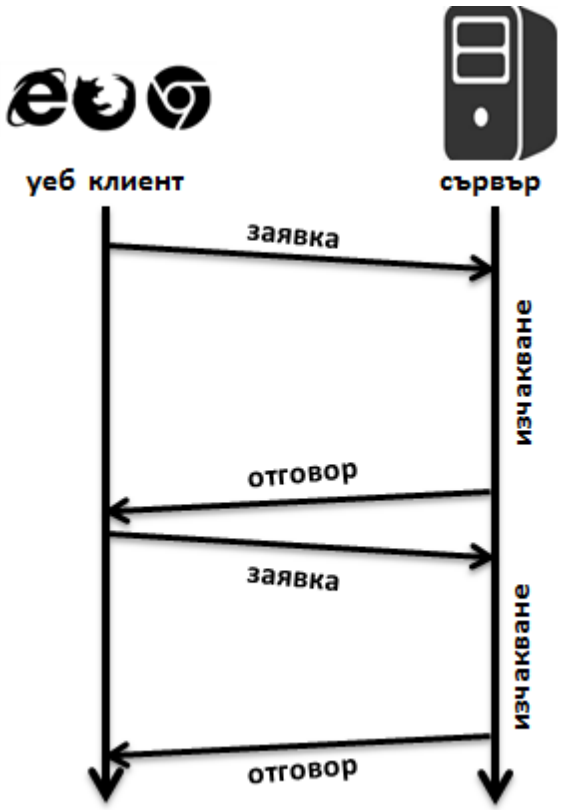
В практиката този подход се реализира чрез използване на поне три фрейма - родителски, видим и невидим. В родителския фрейм се съхраняват шаблоните на уеб страниците и чрез таймер се реализира постоянно презареждане на невидимия фрейм. В него, освен данните, има и кратък програмен код, който се изпълнява веднага след зареждане на страницата и най-често извиква функция от родителския фрейм. Тя, от своя страна, поставя данните между HTML тагове от шаблоните и, или чрез DOM променя само отделни части от вече изобразена уеб страница, или визуализира изцяло генерирана нова уеб страница.

С цел по-нататъшно съкращаване на времевия интервал от момента на настъпване на някакво събитие от страна на сървъра до получаването му от клиента, в практиката са реализирани редица подходи, използващи различни особености при реализацията на протокола HTTP, както при уеб сървърите, така и при уеб клиентите. Подходите, описани по-горе, при които периодично чрез таймер се отправят заявки, в литературата са известни като **"издърпващи" - "pull" техники**, тъй като клиентът периодично "издърпва" нови данни от сървъра и така се създава илюзията за непрекъснат обмен на данни (Prajitno et al., 2019). Както се вижда, те са базирани почти изцяло на HTML и до известна степен на JavaScript.

Следващото стъпало в еволюцията на "pull" техниката се базира на особеност на протокола HTTP по отношение на времето за прекъсване на мрежовата връзка при липса на обмен на данни (timeout). В спецификацията на HTTP/1.0 (Berners-Lee et al., 1996) няма предвидени ограничения за това време, т.е. теоретично мрежовата връзка между клиента и сървъра може да остане отворена неограничен

период от време. На практика обаче масово при реализацията на клиента и сървъра се предвижда такова ограничение, като до около 2010 г. по подразбиране то беше най-често задавано за 300 секунди (5 минути), а в последните години има тенденция тази стойност да намалява до 60 и дори 20 секунди.

Тези особености по отношение на времето за прекъсване се използват за реализация на техника, известна в литературата като **"дълго запитване" - "long polling"** (Loreto et al., 2011). При нея отново периодично клиентът изпраща заявки, но сървърът не връща веднага отговор, а изчаква да настъпи някакво събитие и тогава предава данните за него (фигура 5.2).



Фигура 5.2. Времедиаграма на комуникацията между уеб клиента и уеб сървъра при техниката "дълго запитване" - "long polling".

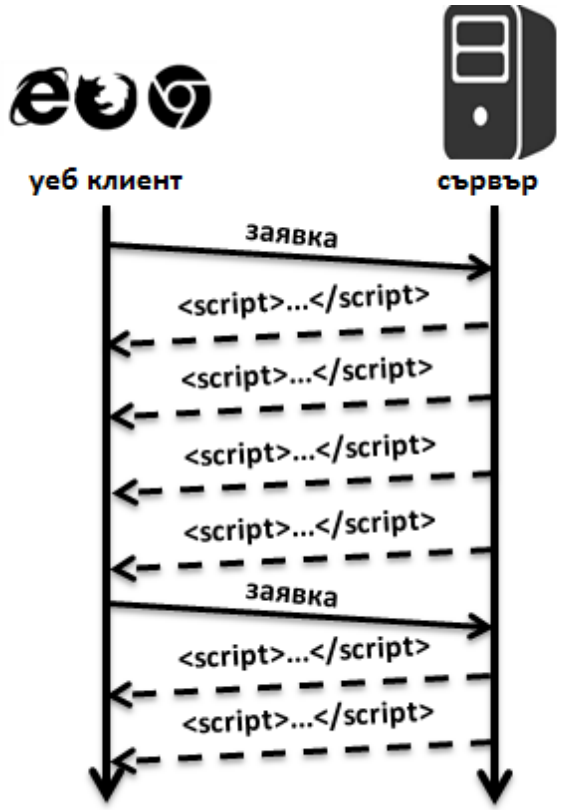
В случай, че наближава времето за прекъсване на мрежовата

връзка при липса на трафик, а събитието все още не е настъпило, се изпраща формален отговор, че няма данни, след което клиентът повтаря процедурата. По този начин мрежовата връзка изкуствено се поддържа отворена продължителен период от време, но за сметка на това веднага при настъпване на събитие сървърното приложение изпраща отговор до клиента. Като допълнителен ефект може да се получи значително намаляване на трафика в сравнение с обикновената "pull" техника, при която сървърът веднага връща отговор, независимо от това дали има нови данни или няма такива. Очевидно е, че намаляване на трафика ще има само когато интервалът от време, през който клиентът изпраща заявките, е по-малък от интервала от време, през който възникват събитията при сървъра.

При "long polling" техниката се предполага, че сървърът не разполага с нужните данни към момента на получаване на заявката и той задържа по-дълго време от нормалното връщането на отговор, докато те не се получат или не настъпи някакво събитие (виж Приложение 2). Когато данните се получат, има два варианта за продължение след като сървърът отговори: или веднага затваря връзката (класически "long polling"), или продължава да я държи отворена и да изпраща данни към клиента като добавя тагове в един "безкраен" iframe (преминаване към "push" техника).

Развитие на "long polling" техниката е т.нар. **"push" - избутваща техника** (в литературата се среща още и като "web page streaming" - Suzumura&Oiki, 2011), при която по отворената мрежова връзка сървърът дълго време изпраща отделни порции физически завършени HTML тагове. Тази техника се базира на факта, че повечето уеб клиенти са така реализирани, че с цел по-бързо визуализиране на уеб страницата, започват интерпретирането на HTML таговете преди да се е получил целия отговор, т.е. преди да се е получила двойката за край на документа </body></html>. Отделните части физически завършени HTML тагове

всъщност представляват програмен код, ограден в тагове <script></script> (фигура 5.3), в който са както данни, така и програмния код (най-често извикване на функции от родителския фрейм), опресняващ съдържанието на видимата уеб страница, фрейм или отделен DOM елемент (виж Приложение 3).



Фигура 5.3. Времедиаграма на комуникацията между уеб клиента и уеб сървъра при техниката "избутване" - "push".

Основен недостатък на тези техники е, че се налага използването на специализирани уеб сървъри (подобни на Node.js), тъй като повечето универсални уеб сървъри (като Apache и IIS) не са предназначени да поддържат голямо количество едновременно отворени мрежови връзки в продължение на дълги периоди от време. Налага се употребата на уеб сървъри с по-малко богатство от възможности, но по-добре използващи оперативната памет за всяка една мрежова връзка.

Допълнителен проблем може да възникне, ако между клиента и сървъра има прокси сървър, чиито настройки да не позволяват мрежовата връзка между тях да остава отворена дълги периоди от време. Възможно е и с цел антивирусна защита прокси сървърът да очаква предаването на цялата заявка или отговор, и след това да я изпрати на другата страна. Заради такива случаи се предпочита използването на HTTPS, тъй като криптираното съдържание обезсмисля неговия междинен анализ за целите на антивирусната защита.

5.3. Съвременни стратегии "AJAX" и "WebSocket"

Както се вижда от гореизложените подходи, чрез усъвършенстване на техниките, използвани при изграждане на уеб приложения, работещи в реално време, подходите по своята същност се различават съществено от класическите разбирания за уеб сайт, доближавайки се като функционалност до десктоп приложенията. След 2005 г. масово започва да се използва вграден обект XMLHttpRequest (Kesteren et al., 2016), като популярността му нараства с широкото използване в уеб услугите на Google - Mail и Maps. Оттогава техниката с използване на новия за времето си обект се нарича AJAX[1] (съкратено от Asynchronous Javascript And Xml[2]).

Технологията AJAX е съвременен начин за създаване на интерактивни разпределени уеб приложения (Петров, 2014b; Петров, 2015c; Kumar et al., 2016). При класическия модел уеб браузърът зарежда цяла уеб страница, изчаква да се случи определено събитие и зарежда друга уеб страница. При използване на AJAX, докато една страница е заредена, на заден план (най-често асинхронно) се изпращат HTTP заявки, без да се презарежда цялата страница. Обработката на HTTP

[1] През последните години все по-често в литературата вместо AJAX се използва съкращението XHR за означаване на обекта XMLHttpRequest.

[2] Това име според нас не отразява точно същността на технологията, тъй като заявките може да се изпращат и синхронно, както и данните може да са в други формати - например TXT, HTML или JSON.

отговора също се извършва на заден план (виж Приложение 4).

Както е известно, действията, които извършва потребителят с мишката или клавиатурата докато работи с една уеб страница, генерират събития. Тези събития могат да се прихващат и да се изпълнява определен програмен код. Прихващането става чрез т.нар. манипулатори, на които се присвоява като низ програмния код, който трябва да се изпълни. Обикновено в низа не се пишат много програмни редове, а се извиква само една функция. Тази функция специално се предвижда да се изпълнява след точно определено действие, като например кликване върху някой елемент, и този начин на програмиране е известен като събитийно програмиране, както вече казахме по-рано.

Манипулаторите, които прихващат събития се записват като атрибути (параметри) на HTML таговете. Различните тагове поддържат различно множество от събития и този събитиен модел е стандартизиран от W3C под формата на технически доклади (Pixley, 1999).

С AJAX сравнително лесно може да се реализират техниките "издърпване" - "pull" и "дълго запитване" - "long polling", но техниката "избутване" - "push" трудно се реализира, като най-често се използват няколко AJAX заявки, работещи в комбинация с техниката "дълго запитване" - "long polling". Популярно средство, с лиценз свободен софтуер, което значително улеснява разработката на уеб приложения в реално време, използващи "push" техниката, е Ajax Push Engine[1]. За съжаление сървърната част изисква операционни система UNIX/Linux/MacOSX, тъй като използваните техники за оптимално поддържане на голям брой мрежови връзки не работят под Windows. Друго популярно средство, специално за разработчици на Java, е CometD[2], също с лиценз свободен софтуер.

При класическите заявки по протокол HTTP 1.0, уеб клиентът

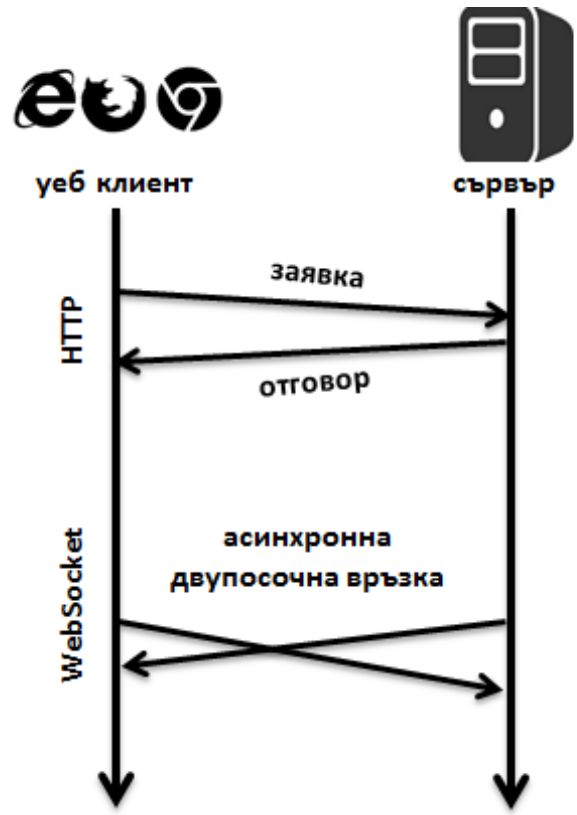
[1] Виж: (Ape-project.org, 2019).
[2] Виж: (Cometd.org, 2018).

изпраща заявка до уеб сървъра, получава отговор и мрежовата връзка между двете страни се прекъсва. Версия 1.1 на HTTP даде възможност в рамките на една мрежова връзка уеб клиентът синхронно да изпраща на уеб сървъра множество заявки, като по този начин се избягва т.нар. "трипасово ръкостискане" (характерно за TCP/IP) при отваряне на една нова мрежова връзка (сокет), както и разходите от време, свързани със затварянето й. В резултат, мрежовата връзка между клиент и сървър стои отворена много по-дълго време, спрямо връзките по протокол HTTP 1.0. Това наложи еволюцията на уеб сървърите да се развие и в посока поддържане на по-голям брой едновременно отворени мрежови връзки, с цел да няма отказ от обслужване при голям брой HTTP заявки за единица време. В резултат се създават и се налагат на пазара уеб сървъри, способни да поддържат десетки хиляди едновременно отворени връзки при сравнително малък обем заета оперативна памет, като например nginx и lighttpd. За сравнение, при другите по-популярни уеб сървъри - Apache и IIS, обемът памет, заеман за всяка мрежова връзка е много по-голям, което налага и по-големи изисквания към хардуера по отношение на инсталираната оперативна памет.

От друга страна, реализацията на техниките "издърпване" - "pull", "дълго запитване" - "long polling" и "избутване" - "push" в повечето случаи изисква използването на средства, които по принцип отговарят на стандартите, но не са изрично предвидени в тях. Затова през последните години за създаването на приложения в реално време се предпочитаха други подходи, като Java аплети, Flash или Silverlight приложения. Използването на подобни приложения затруднява потребителите, тъй като изисква инсталирането на допълнителни модули (плъгини) към браузърите и това намалява тяхната приложимост.

В документния обектен модел на уеб страниците (DOM) през последните години бяха предложени за добавяне няколко глобални

обекти, специално предназначени за разработката на уеб приложения в реално време. Такива обекти са EventSource, използващ протокола Server-Sent Events (ще бъде разгледан в следващата точка) и обектът **WebSocket**, стандартизирани от W3 Consortium (Hickson, 2012). За WebSocket има и специално разработен протокол от IETF със същото име (Fette & Melnikov, 2011), поддържан от най-популярните браузъри към настоящия момент. Това осигурява широкото разпространение на WebSocket и той е особено перспективен, тъй като решава всички въпроси, свързани с двупосочната асинхронна връзка между клиента и сървъра (фигура 5.4). Той опростява значително създаването на уеб приложения в реално време, като отпадат проблемите, налични при гореописаните подходи по отношение на множеството от допълнителни програмни библиотеки (Петров, 2014а).



Фигура 5.4. Времедиаграма на комуникацията между уеб клиента и уеб сървъра при HTTP и WebSocket.

При някои изследвания се установява, че след замяната на AJAX с WebSocket мрежовия трафик спада с около 50%. Друга установена полза от същите автори е, че заемането на оперативна памет при използване на WebSocket се извършва с равномерни темпове, а не рязко, както е при използването на AJAX (Puranik et al., 2013).

Редица големи доставчици на облачни услуги поддържат WebSocket (Cloud.google.com, 2017; Munns, 2018; Schackow, 2013; Gupta, 2017), което дава възможност за прилагането му и без пълен административен контрол върху хардуерния сървър при хостинг на такива приложения. Възможността по ефективен начин (по отношение на обем трафик и натоварване на сървъра) да се предават данни на малки части, с висока честота и асинхронно, създава предпоставки за разработването на приложения, които работят също толкова бързо и интерактивно, както и традиционните десктоп приложения.

Подходите, базирани изцяло на особеностите на протокола HTTP, не са подходящи за тези цели, тъй като при предаването на заявката и на отговора към данните се добавят и заглавни части, които съдържат различна служебна информация, като тази за типа на данните, кодировка, бисквитки, идентификация на клиента или сървъра, дата, контролни суми и др. Тези служебни данни варират значително по обем и съпоставени с размера на една уеб страница или изображение са сравнително малки, но когато данните се изпращат на малки порции и през малък интервал от време, обема на служебните данни започва да става съпоставим и може в пъти да надхвърли обема на "полезните данни", които всъщност единствено трябва да се предават.

За разлика от сесийния синхронен характер на HTTP (заявка - отговор), при WebSocket, след като веднъж се осъществи връзката, двете страни са равноправни и използват постоянна мрежова връзка. Обменът на данни е асинхронен, което означава, че двете страни на мрежовата връзка не е нужно да се изчакват, за да изпращат данните и това може да

става едновременно и от двете страни. Използването на събития от страна на клиента дава възможност без да се прекъсва изпълнението на програмен код при него, след пристигане на данни от сървъра те да бъдат обработени.

Едно от основните предимства на WebSocket е опростения приложен програмен интерфейс, който заедно с голямото количество библиотеки, предназначени да улеснят създаването на уеб приложения в реално време, значително подпомага разработката от страна на уеб клиента (виж Приложение 5).

Работата от страна на сървъра е по-сложна и изисква повече познания за стандартите на WebSocket. Затова при програмиране на Node.js с използване на WebSocket и за клиентската, и за сървърната част може да се използва програмната библиотека Socket.IO (Socket.IO, 2019; Mardan, 2018).

Независимо дали ще се използва AJAX или WebSocket основен проблем при работата в режим реално време представлява т.нар. "латенция" на мрежовата връзка, с която се означава времезакъснението при изпълнение на дадена операция. По отношение на компютърните мрежи латенцията включва в себе си времезабавянето, което възниква по време на предаване на данните, тяхното обработване и представяне за крайния потребител. Често латенцията още се нарича и лаг, когато се отнася до цялостното субективно усещане за времезабавянето поради различни причини при интерактивна работа със система, работеща в режим онлайн (Earnshaw et al., 2012, p.145).

В зависимост от същността на уеб приложението, под латенция може да се разбира времезабавянето само в едната посока - от момента на генериране на съобщението при източника до момента на неговото получаване, но също така под латенция може да се разбира времезабавянето и в двете посоки на комуникационната връзка - в този случай включва времето от изпращане на първоначалната заявка,

нейната обработка от отсрещната страна, генериране на отговор и последващото му получаване от първата страна. При втория вариант латенцията е около два пъти по-голяма спрямо първия вариант.

За лага допринасят множество независещи един от друг фактори и той може да се представи като сума от:

- времето за подготовка на съобщението от страна на клиента плюс времето необходимо на сигнала да измине пътя през физическата медия (оптични кабели, радиовълни, електрически сигнали и прочие);

- времето за обработка на сигнала в междинните устройства;

- времето, необходимо за обработка на съобщението от страна на сървъра, генериране на отговор, изпращането му обратно на клиента;

- и отново в обратна посока - времето за изминаване на пътя от сигнала във физическата медия;

- времето за обработка в междинните устройства;

- времето необходимо на клиента да обработи и евентуално да визуализира отговора.

В компютърна мрежа, в която има относително малък трафик, който се явява част от максималната пропускателна способност на средата и ако приемем, че настройките на междинните устройства и настройките от страна на клиента и сървъра са извършени правилно (оптимално), то в този случай определящо за големината на латенцията има скоростта на разпространение на сигнала, която е ограничена от физичните закони - скоростта на разпространение на светлината във вакуум (около 300 хил. км/с).

В среда на локална мрежа, където клиентът, сървърът и междинните устройства са сравнително близко разположени едни до други, и при сравнително прости заявки, времезабавянето може да спадне до 1-2 мс.

В среда на глобална мрежа, където клиентът и сървърът са сравнително на големи разстояния един от друг, при същия тип прости

заявки времезабавянето може да надхвърли 500 мс, ако например комуникационната връзка се извършва с помощта на спътник, разположен на геостационарна орбита. В този случай сигналът на заявката-отговор трябва да измине 4 пъти разстоянието от Земята до спътника на геостационарна орбита (около 36 хил. км. над Екватора), при което времезабавянето от междинните устройства става пренебрежимо по-малко в сравнение с времето за пренасяне на сигнала, ограничено от законите на физиката.

Не по този начин стои въпросът, когато в компютърната мрежа има относително голям трафик, който се доближава до максималната пропускателна способност на средата, пренасяща сигнала. В подобни случаи междинните мрежови устройства започват да буферират TCP/IP пакетите или да търсят други маршрути, при което част от данните може да се загубят, което да наложи тяхното повторно изпращане. Тогава основен дял за латенцията започва да играе друг фактор - времето за обработка в междинните устройства и делът на загубените TCP/IP пакети.

Освен разстоянието и натоварването на междинните мрежови устройства, голямо влияние за латенцията оказват също така клиентът или сървърът. Ако към сървъра се отправят заявки, които изискват по-продължителен период от време за тяхната обработка или ако от страна на сървъра се изисква повече време за интерпретиране на отговора, то това също може да доведе до увеличаване на латенцията, а от там и на лага.

Особено неблагоприятно влияние за лага, като едно субективно усещане за времезабавянето, се оказва загубата на TCP/IP пакети. Тази загуба може да се получи поради най-различни причини и може да няма пряко отношение към латенцията, но независимо поради какви причини един пакет с данни е загубен, то неговото повторно изискване, изпращане и получаване в крайна сметка оказва негативно влияние

върху завършеността на конкретната операция и в резултат субективното усещане на потребителя за качеството на работа в режим онлайн се понижава. В подобни случаи, когато се елиминира причината за загуба на пакети, лагът се намалява.

Можем да обобщим, че определящи фактори за лага са: пропускателната способност на компютърната мрежа, нейното моментно натоварване, разстоянието между двете страни на комуникационната връзка, наличието на загуби на пакети, натоварването и възможността за обработка на данни от страна на сървъра и от страна на клиента. Тези фактори следва да се имат предвид при разработката на уеб приложения, работещи в реално време.

5.4. Стратегии за използване в перспектива "SSE" и "HTTP/2"

Като заместител на все по-усложняващите се техники за реализация на "push" технологията е създаден протокола **Server-Sent Events (SSE)**, който е одобрен от организацията W3C (Hickson, 2015). Протоколът към началото на 2019 г. се поддържа от съвременните версии на Firefox, Chrome, Opera, Safari и др., но не се поддържа от Internet Explorer 11 и Edge 18 - браузърът на Windows 10. Според публикация на вицепрезидента на Microsoft следващите версии на Edge ще са базирани на платформата Chromium (Belfiore, 2018), разработена от Google и съответно ще поддържат SSE.

При SSE, уеб приложение чрез изпращане на заявка се "абонира" към поток от събития, генерирани от уеб сървър. Връзката остава отворена неограничен период от време и когато ново събитие се появи при сървъра, към клиента се изпраща съобщение по нея. Връзката е основно еднопосочна - от уеб сървъра към уеб клиента и SSE на практика представлява стандартизиран и усъвършенстван вариант на "long polling" и "push" техниките (виж Приложение 6).

Предполагаме, че този протокол не получава подкрепа от Microsoft, тъй като протоколи като WebSocket предлагат повече възможности - двупосочна асинхронна връзка. Въпреки че използването на двупосочен канал за връзка дава повече възможности пред еднопосочния (при създаването на игри, приложения за съобщения и прочие), съществуват различни ситуации, при които не се налага активно да се изпращат данни от клиента към сървъра. Например, следене на обновяване на потребителски статуси, новини, борсови котировки и т.н. В случай, че се налага изпращането на данни от уеб клиента до уеб сървъра, може да се изпрати отделна заявка чрез AJAX или по друг начин.

Основният проблем на протоколите HTTP 0.9/1.0/1.1 е, че те са синхронни. При тях се изисква клиентът да изчаква сървъра да върне целия отговор, след което да се изпрати нова заявка. При този начин на работа, ако сървърът по-бавно генерира някакъв ресурс, това на практика блокира цялата последваща комуникация, а обикновено за визуализирането на една уеб страница са необходими множество отделни ресурси - изображения, скриптове, стилове, шрифтове, фреймове и т.н. За преодоляване на този проблем повечето браузъри отварят едновременно няколко мрежови връзки към сървъра, с помощта на които могат да получават няколко ресурса едновременно (например Google Chrome отваря едновременно 6 мрежови връзки). Това ограничение до няколко връзки се отнася за едно име на домейн и поради тази причина в един момент от време се популяризира тактиката част от ресурсите да се разполагат на други домейни с цел да се позволи да се използват едновременно повече мрежови връзки за цялостно зареждане на страниците (това е т.нар. подход за "domain sharding"). Друга тактика е малките файлове от един и същи тип да се обединят в един голям файл, така че да се минимизира изчакването, получаващо се в резултат на множество последователни двойки заявка-отговори.

Например малките изображения се обединяват в така наречените спрайтове ("CSS image sprites"). За решаването на подобни проблеми през 2009 г. Google предлага нов протокол, наречен SPDY, който след това се трансформира в протокола HTTP/2.

Както се посочва от някои автори (Belshe et al., 2015), протоколът HTTP/2 се различава съществено от HTTP 0.9/1.0/1.1 - той не е синхронен, а асинхронен; не е текстов, а двоичен; използва само една TCP/IP връзка за домейн, през която се извършва мултиплексиране, т.е. предаване на множество потоци данни едновременно. Последното се реализира, като всяка двойка заявка-отговор се асоциира със свой собствен поток и съответно данните, които трябва да се изпратят, се разделят в отделни фреймове[1]. Фреймовете се асоциират с определени потоци и по този начин през една мрежова връзка може да се предават множество потоци от данни. Потоците са относително независими едни от други, така че блокирането на някой отговор или заявка не пречи на работата на останалите потоци. По този начин множество заявки-отговори могат да бъдат изпращани едновременно и което е по-важно - потоците могат да бъдат приоритизирани. Мултиплексирането намалява необходимостта да се разполагат ресурсите на различни домейни (подхода "domain sharding") или да се използват т.нар. "спрайтове".

Протоколът HTTP/2 поддържа възможност за компресиране на заглавните части. Това не се извършва чрез класическите алгоритми за компресиране на данни, а чрез специфична организационна техника, използването на която осигурява да няма повторно изпращане на полета от заглавните части, които вече са били изпратени. За целта и клиентът, и сървърът поддържат таблици с изпратените и съответно получените полета в заглавните части, както и техните стойности. При първата двойка заявка-отговор се изпраща пълната заглавна част, а при

[1] В HTTP/2 са предвидени различни видове фреймове за различни цели например HEADERS, DATA, GOAWAY, PRIORITY, SETTINGS, PUSH_PROMISE и др.

последващи заявки-отговори дублиращите се полета се пропускат. Икономията на трафик достига средно до 0,5KB при някои заявки-отговори и това се оказва много ефективен механизъм за значително намаляване на размера на заглавната част.

Друга важна особеност на HTTP/2 е възможността сървърът да изпраща ресурси, които не са били изрично заявени от клиента - това е т. нар. "техника на избутване" (server push)[1]. Тези ресурси се кешират от страна на клиента за бъдеща употреба. Сървърът може да започне да изпраща тези ресурси непосредствено след като връзката е била установена, без дори да чака клиентът да изпрати заявка. При този начин на работа е възможно да се увеличи ненужно мрежовия трафик, но като цяло уеб страниците се зареждат по-бързо.

Въпреки, че стандартът не го изисква, практическата реализация на поддръжката на HTTP/2 в браузърите налага използването му задължително в комбинация с HTTPS.

С цел емпирично изследване на намаляването на лага при зареждане на уеб страници чрез използване на различни протоколи HTTP - версии 1.0, 1.1 и 2, създадохме опитна установка, чрез която да установим лага при различни стойности на мрежова латенция. Важна особеност при използване на HTTP/2 е, че съвременните браузъри поддържат версия 2 само, ако връзката е криптирана, въпреки че това не се изисква изрично по стандарта. Затова реално използвахме HTTPS/2 и допълнително добавихме изследване за HTTPS/1.1.

Тъй като клиентът и сървърът физически са разположени на една машина[2], времезабавянето (латенцията) е минимално. С цел симулиране на мрежова латенция би могло да се използва допълнителен софтуер, с чиято помощ се задържат за определен период от време изпращаните и

[1]. За пример виж (Peon&Ruellan, 2015; Bach et al., 2017).

[2] При проведените тестове са използвани компоненти със следните характеристики: операционна система - Windows 7, 32 bits; процесор - i5-2430M; оперативна памет - 4GB; уеб сървър - Apache/2.4.18.

пристигащи TCP/IP пакети и по този начин може да се симулира териториална отдалеченост на двете страни на мрежовата връзка. Разбира се, при използване на реална глобална мрежа, латенцията би варирала постоянно, тъй като маршрутът, по който се движат пакетите, може динамично да се променя.

Променящият се маршрут оказва влияние както на броя на устройствата, през които преминават пакетите в двете посоки, така и на разстоянието, което изминава сигнала. Дори и маршрута, по който се движат пакетите, да не се променя, е възможно отделни мрежови устройства да са подложени на пиково натоварване, което да забавя моментно тяхната работа. Очевидно е, че ако се симулира латенция, то тези фактори са премахнати.

При настройване на уеб сървъра за работа с HTTP/2 са следвани указанията от документацията на Apache (Httpd.apache.org, 2018b), а именно - в конфигурационния файл httpd.conf е активирана директивата за зареждане на mod_http2, добавена е директива за превключване от версия 1.1 към версия 2:

LoadModule http2_module modules/mod_http2.so

Protocols h2 h2c http/1.1

Тъй като в хода на работа се установи, че браузърът Mozilla Firefox не превключва автоматично към използване от по-ниска на по-висока версия, се наложи допълнително да се променят две директиви за работа с SSL - директивите SSLCipherSuite и SSLProtocol.

При настройване на уеб сървъра за работа с HTTP/1.0, отново са следвани указанията от документацията на Apache (Httpd.apache.org, 2018a) и са добавени директивите:

SetEnv downgrade-1.0

SetEnv force-response-1.0

Тези настройки бяха включвани и изключвани при необходимост, тъй като по подразбиране уеб сървърът използва версия 1.1 и при HTTP,

и при HTTPS. При провеждане на тестовете се спазва следния подход: уеб сървърът се спира; извършват се съответните настройки и пак се стартира; на програмата за симулиране на латенцията се задават съответни параметри; три пъти подред се зарежда ресурса с цел "подгряване" на системата и след това в нови "инкогнито" прозорци се извършва теста последователно в двата браузъра по три пъти, след което времето се осреднява за всеки един от тях.

Програмният модул, който генерира уеб страница, която от своя страна изисква зареждането на допълнителни ресурси, се изпълнява от интерпретатор на PHP 7 и съдържанието му е показано по-долу. Модулът се използва за определяне скоростта на зареждане на уеб страници при използване на различни протоколи.

```
<?php
	$files = 107;
	$size = 32; //KB
	$latency = 0; // milliseconds
	if(!isset($_GET['t'])) {
		print<<<EOT
<html>
<head>
  <script>
	var begin=0,
	end=0,
	delta = 0;
	function print(s) {
		document.getElementById("txt").innerHTML +=
			(Date.now()) + ":" + s + "<br />";
	}
	document.addEventListener(
		"DOMContentLoaded",
```

```
            function(event) {
                begin = Date.now();
                print("DOMContentLoaded");
            }
        );
        window.onload = function () {
                end = Date.now();
                print("load");
                print("delta="+(end-begin));
        }
    </script>
</head>
<body>
<div id="txt"></div>
EOT;
        usleep($latency*1000);
        echo "<script>print('START: $files files, $size KB');</script>\n";
        for($i=0; $i<$files; $i++)
            echo "<script src='?t=$i' async></script>\n";
        print<<<EOT
</body>
</html>
EOT;
        } else {
            usleep($latency*1000);
            echo "print('$_GET[t]'); tmp = '" . str_repeat("1234567890", $size*100) . "';";
        }
?>
```

В променливите $files и $size се задават съответно колко допълнителни ресурси трябва да се заредят за цялостното визуализиране

на уеб страницата и какъв да бъде техния размер. Допълнително в променливата $latency може да се зададе стойност за времезабавяне в милисекунди, което да симулира извършването на повече обработки от страна на сървъра при връщането на ресурса или неговото по-голямо натоварване от други клиенти. Подобно времезабавяне реално ще е налице например при връзка със сървър на бази от данни.

При провеждането на тестовете са зададени такива стойности на променливите, които максимално близко да възпроизведат средната уеб страница към момента на провеждане на тестовете, съгласно статистически данни от авторитетни изследвания (Httparchive.org, 2019), а именно - уеб страница, която зарежда допълнително 107 ресурса с общ обем 3,5MB.

В таблица 5.1 е показан интервалът от време в секунди от момента на зареждане на една средно статистическа уеб страницата до момента на зареждане на всички ресурси, необходими за визуализирането й (лаг) при използване на различни уеб протоколи. При подобни експерименти трябва да се отчита ролята и значението на хардуерната производителност (Sulov, 2014).

Таблица 5.1. Лаг в зависимост от използвания протокол при работа с браузъри Google Chrome и Mozilla Firefox

протокол	**браузър**	
	Google Chrome 63.0.3239	**Mozilla Firefox 57.0.4**
HTTP/1.0	0,5 s	0,5 s
HTTP/1.1	0,4 s	0,6 s
HTTPS/1.1	0,5 s	0,8 s
HTTPS/2	0,3 s	0,3 s

Според нас, при представените данни от съществено значение не са конкретните абсолютни стойности, а по-скоро съотношенията между тях. В някои от тестовете по-добре се представя единият браузър, а при други - другия и това също не е от съществено значение. Резултатите от тестовете показват подобряване, в смисъл намаляване на лага, при използване на HTTP/2. Забелязва се, че като цяло използването на HTTPS/1.1 увеличава лага и това е напълно естествено, тъй като за криптиране и декриптиране се използват допълнителни изчислителни операции, които изискват ресурси и отнемат повече време. На практика, използването на сигурна криптирана връзка се явява едно допълнително междинно звено при комуникацията, което внася допълнително времезабавяне. Извършеният експеримент няма претенциите за всестранно и задълбочено тестване на отделните протоколи, но показва, че дори и "механичното превключване" към използване на HTTPS/2 носи ползи за намаляване на лага.

ЗАКЛЮЧЕНИЕ

В банковата система, както е известно, работи високо квалифициран технически персонал, който отговаря за ИТ инфраструктурата. Основен компонент в последната е уеб сайтът на банката, който в последните години се превърна в основно средство за взаимодействие с клиентите. Характерно за банковата система е, че решенията обикновено се вземат на база анализ на добри практики с цел да се намали риска и в повечето случаи липсва недостиг на финансиране, което дава възможности цената да не се разглежда като основен фактор при избора на софтуер.

Уеб страницата в съвременен вариант представлява мултимедийно интерактивно представяне на информация в електронен вид, като чрез хипервръзки е възможно да се свързват различни информационни елементи.

Сигурността при използване на HTTPS се постига в няколко различни направления: осигурява се автентификация на сървъра посредством цифрови сертификати, издавани от доверени сертифициращи органи; евентуално сървърът може да извършва автентификация на клиента, ако той също има сертификат; осигурява се високо ниво на поверителност на обменяните данни между двете страни, като данните се криптират; криптирането допълнително осигурява и интегритета на обменяните данни, т.е. гарантира се тяхната цялостност и не е възможно незабелязано да се подменят част от тях.

В последните години големи организации и компании, като Electronic Frontier Foundation, Mozilla, Akamai, Cisco, IdenTrust и др., създадоха сертифициращ орган, който да издава безплатно сертификати. Тези сертификати засега са с период на валидност от 90 дни. От началото на 2018 г. се очаква да започне да се предлагат и т.нар. wildcard сертификати, покриващи всички поддомейни на един домейн. По всяка вероятност това ще окаже сериозно влияние за масово преминаване към

използване на HTTPS по света, включително и от българските банки, които все още не използват този протокол.

От данните за естонските банки, събрани през април 2018 г. и през август 2019 г. се установи, че средният период на валидност на SSL/TSL сертификатите, използвани от банките, се повишава от 15,4 месеца на 17,4 месеца. Появява се нов пазарен играч на естонския банков сегмент - Sectigo (бивше Comodo преди 2018 г.), чиито сертификати започват да се използват от 2 банки към август 2019 г. Като цяло се установи засилване на консолидацията на пазара на SSL/TSL сертификати в Естония - увеличава се пазарния дял на DigiCert, което отчасти се дължи на обстоятелството, че от 2017 г. компанията Thoma Bravo е собственик на DigiCert, Symantec, Thawte и GeoTrust. Това дава възможност за прилагане на обща координирана политика по отношение на издаването на SSL/TLS сертификати. Последното наблюдение важи и за други държави.

Като цяло за балтийските банки се установи, че мнозинството - 15 балтийски банки, използват обикновени DV сертификати, а само 10 балтийски банки използват по-сложните за издаване на EV, които са и по-скъпи. Само в Естония мнозинството (56%) от банките използват сертификати за EV, докато в Латвия и Литва ситуацията е обратна - мнозинството (57% в Латвия и 75% в Литва) използват DV. Средната валидност на сертификатите е 19 месеца, с медиана 23 месеца. Изследването, проведено върху използването на протокола HTTPS на публичните уебсайтове на балтийските банки, обхвана сайтовете на всички 9 естонски, 14 латвийски и 4 литовски банки, лицензирани да оперират на съответната територия на страната от местните национални банки или друга държавна институция.

Централно европейските държави Чехия, Словакия и Унгария споделят много прилики в демографски и икономически план, но от гледна точка на паричната политика, има съществени различия.

Словакия използва евро, а Чехия и Унгария използват свои собствени валути. По отношение на икономическото си развитие на база индикаторите БВП на глава от населението, БНД на глава от населението, HDI и кредитни рейтинги държавите се подреждат от най-развита към по-слабо развита в следния ред: Чехия, Словакия и Унгария.

Установихме, че към август 2019 г. в Чешката република по-голямата част от банките - 77,3%, използват EV сертификати; в Словакия – 40% от банките използват EV; и в Унгария – 31,8% от банките използват EV. Забелязва се наличието на пряка положителна връзка между нивото на икономическо развитие и използването на по-сложните за издаване и по-скъпи сертификати EV.

Установихме, че в Чехия никоя от банките не използва безплатни SSL/TLS сертификати. В Словакия 1 банка използва безплатен SSL/TLS сертификат. В Унгария 3 банки използват безплатни SSL/TLS сертификати. Забелязва се пряка отрицателна връзка между нивото на икономическо развитие и използването на безплатни SSL/TLS сертификати.

При средният период на валидност на SSL/TLS сертификатите също се забелязва пряка отрицателна връзка - в Чехия е 20 месеца, в Словакия е 19 месеца и в Унгария е 18 месеца.

По отношение на банки от държави на Балканския полуостров изследването обхвана публичните уеб сайтове на всички 20 български, 24 румънски и 27 сръбски банки, лицензирани да оперират на територията на съответната държава от местните централни банки.

Ситуацията в България беше по-подробно изследвана в продължение на 3 години - в периода юли 2016 - март 2019 г. Към юли 2016 г. от софтуера за уеб сървъри най-използван е Apache - 55%, следван от IIS и nginx с 14%. Само 27% имат защита срещу Clickjacking атаки чрез използване на полето X-Frame-Options. Езикът за програмиране, използван за динамично генериране на съдържанието,

най-често е PHP - 50%. Използват се готови системи за управление на съдържанието Joomla, Wordpress и Orchard. Установи се, че библиотеката jQuery и/или jQuery-UI се използва в 82% от банковите уеб сайтове, Modernizr в 27% от тях. Много от сайтовете използват по няколко платформи едновременно за аналитично следене на ползваемостта: Google Analytics - 91%, Facebook Retargeting - 27%, DoubleClick.Net - 23%, Google Tag Manager - 23%, Google AdWords Conversion Tracking - 18%, Google Remarketing - 18%.

При анализ на данните, събрани последователно през интервал от 16 месеца - през юли 2016 г., ноември 2017 г. и март 2019 г., се установи, че делът на банките, използващи HTTPS без проблеми, варира за разглеждания период, но като цяло отбелязва положителна тенденция: 59% към юли 2016 г., 50% към ноември 2017 г. и 90% към март 2019 г. Делът на EV сертификатите расте: от 55% към ноември 2017 г. на 72% към март 2019 г.

При сравнение на данните към март 2019 г. за България с тези за Румъния и Сърбия се установи, че 90% от българските, 96% от румънските и 89% от сръбските банкови уеб сайта използват HTTPS без съществени проблеми. В България мнозинството (13 банки) използват EV, а останалите (5 банки) използват DV. В Румъния ситуацията е сходна - мнозинството (13 банки) използва EV, а останалите (10 банки) използват DV. Но в Сърбия е обратното - мнозинството (18 банки) използват DV, а останалите (6 банки) използват EV. При два случая в България, един случай в Румъния и четири случая в Сърбия добрите практики не се следват от банковите уеб сайтове и те не пренасочват автоматично от несигурна HTTP към сигурна HTTPS връзка.

С най-голяма популярност сред сертифициращите организации в България се ползват: Comodo и GeoTrust - по 6 банки, следвани от DigiCert - 4 банки. В Румъния най-популярната сертифицираща организация е DigiCert - 12 банки, последвана от GeoTrust - 5 банки. В

Сърбия най-популярните сертифициращи организации са други участници на пазара, които не са представени в България и са слабо представени в Румъния: Thawte - 6 банки, cPanel - 4 банки, Go Daddy - 3 банки. Единственият голям общ играч е GeoTrust - 4 банки.

Една банка в България и по две банки в Румъния и Сърбия използват безплатни 3-месечни сертификати от Let's Encrypt.

Резултатите от проучването биха могли да имат важно практическа полза за управителите на банки и ИТ специалистите при оценяването на технологичните варианти, които трябва да се използват, за да се сведе до минимум рискът за финансовата институция. Също така резултатите разкриват някои добри практики, използвани в банките.

При работа в режим реално време е необходимо да се избере стратегия, при която събирането и обработката на данни от сървърната страна на разпределеното приложение ще достигнат до своя потребител. Необходимо е да се осигури както непрекъснатост на информационния поток между сървъра и клиента, така и да се отчита изискването, че интервалът от време за реакция (времезабавянето) трябва да е колкото се може по-малък. За целта при техническата реализация могат да се използват няколко различни стратегии, които можем условно да групираме в три основни категории: класически стратегии, съвременни стратегии и стратегии в перспектива.

Разликата между отделните категории технологични стратегии е, че първата има универсален характер и може да се прилага за широк кръг от устройства и браузъри, но ефективността по отношение на изчислителни ресурси и обем трафик не е добра. При втората категория стратегии се използват съвременни и ефективни технологии, но все още има в употреба устройства и браузъри, които не ги поддържат - това са т.нар "морално остарели" устройства. Третата категория стратегии все още е технологично неприложима за голяма част от употребяваните устройства и браузъри, но очакваме това да се промени в бъдеще.

Направени бяха тестове за определяне на интервала от време от момента на зареждане на една средно статистическа уеб страницата до момента на зареждане на всички ресурси, необходими за визуализирането й (лаг) при използване на различни уеб протоколи. Резултатите от тестовете показват подобряване, в смисъл намаляване на лага, при използване на HTTP/2 спрямо други варианти.

Използването на платформата Node.js осигурява редица предимства при разработката на информационни системи за работа в реално време - асинхронно (неблокиращо) изпълнение на функции и поддръжка на събитийно-ориентираното програмиране.

Проведени бяха и тестове за определяне на скоростта за преобразуване на JSON низ в програмен обект при използване на различни подходи и използване на различни браузъри. Резултатите от тях показват, че в общия случай трябва да се предпочита използването на подход "JSON.parse(str)". Скоростта на обработка на JSON низове при него е средно около 2-3 пъти по-висока спрямо останалите подходи.

При големи обеми данни скоростта при обработване на заявките понякога се явява от висока степен на важност за работа в реално време и минимизирането на времето за отговор може да се реализира посредством използването на структура от тип кеш. Би могло резултатите от бавно изпълняваните SQL заявки да се съхраняват в междинно хранилище с многократно по-малко време за отговор от тип NoSQL, в което да се намира определено подмножество от всички данни в базата от данни.

От проведените тестове се установи, че MySQL е десетки пъти по-бавен при запис на данни, спрямо Redis, като при увеличаване на броя записи в таблицата, тази разлика се увеличава. Долният диапазон за относителна производителност при запис MySQL:Redis е около 1:30 - 1:40, като при усложняване на структурата на добавяните записи (брой колони, тип данни, индексни полета и т.н.) този показател ще се влошава

за MySQL, т.е. ще е над 1:40. Това се дължи както на сигурния начин, по който се изпълняват SQL-заявките, така и на богатите възможности на релационния модел.

Долният диапазон за относителна производителност при четене MySQL:Redis е около 1:3. И при двете системи се забелязва, че обемът на съхраняваните данни почти не оказва влияние върху времето за търсене на нужния запис или ключ, т.е. времето за търсене е почти постоянно. При увеличаване на сложността на заявките чрез използване на пълните възможности на SQL, този показател отново ще се влошава за MySQL, при което съотношението ще бъде над 1:3.

При положение, че записването на данни в Redis отнема незначително време спрямо подобна операция в MySQL, добавянето на кеширащ слой, използващ NoSQL в едно приложение, е обосновано при определена степен на вероятност данните да се намират в кеша и да не се налага изпращане на заявка до СУРБД. В този конкретен случай полза ще има при съотношение 1 запис, последван от поне 2,5 четения, т.е. при вероятност за намиране на данни в кеша над 60%.

Очертани са рамките при използването на различни формати и като особено подходящи за динамичен обмен на данни в реално време между клиента и сървъра се предлагат форматите JSON и XML. При оперативното съхраняване на данни от страна на сървъра NoSQL системата Redis предлага сравнително добра производителност. Реализацията на асинхронност при различни операции при Node.js го прави подходящ за широк кръг от задачи при работа в реално време. Поради тази причина в голяма част от примерите, дадени в приложенията, се използва именно тази мултиплатформена среда за разработка на сървърни приложения.

ПРИНОСИ НА ДИСЕРТАЦИОННОТО ИЗСЛЕДВАНЕ

Теоретичната и практическа значимост на труда и неговите основни приноси се изразяват в следното:

А. Научни приноси с теоретичен характер

1. Изследвано е в исторически план развитието на основните компоненти на уеб технологиите, софтуерни средства за реализация на уеб технологиите, криптиране на уеб трафика и на тази база са очертани насоките в развитието на уеб технологиите.

2. Разработени са въпросите за методологията на управление на финтех организациите и някои стратегически проблеми във връзка с управление приложението на информационните технологии

3. Изследвани и анализирани са различни подходи за съхраняване на данни с оглед тяхната ефективна обработка в режим реално време, както и са изследвани варианти за работа на уеб приложенията в режим реално време с оглед подобряване на производителността.

Б. Научни приноси с практико-приложен характер

4. Изследвани са уеб технологиите, прилагани в съвременни публично достъпни банкови уеб сайтове на регионален принцип - балтийски държави, централноевропейски държави и балкански държави. Чрез прилагане на сравнителен анализ между съседни държави са установени зависимости в практиките на банките в отделните държави по отношение на средствата за сигурност на техните публични уеб сайтове.

5. Разработени са и са експериментирани алгоритми за оценка на влиянието за повишаване на производителността в следните три направления: кеширане на отговор от SQL заявки с NoSQL система, варианти за обработка на данни във формат JSON и времезабавяне (лаг) при използване на различни версии на уеб протокол HTTP.

ИЗПОЛЗВАНА И ЦИТИРАНА ЛИТЕРАТУРА

1. Амор, Д. (2000). (Р)еволюцията на е-бизнеса. София:ИнфоДАР.
2. БНБ (2016). Списък на лицензираните банки и клонове на чуждестранни банки в Република България, 25 март 2016 г. [Online] Available at: http://www.bnb.bg/BankSupervision/BSCreditInstitution/BSCIRegistrers/BS_CI_REG_BANKSLIST_BG [Accessed 09/08/2016].
3. БНБ (2017). Списък на лицензираните банки и клонове на чуждестранни банки в Република България, раздел I. Банки, лицензирани в Република България, 12 юли 2017 г., [Online] Available at: http://www.bnb.bg/BankSupervision/BSCreditInstitution/BSCIRegisters/BS_CI_REG_BANKSLIST_BG [Accessed 11/11/2017].
4. БНБ, (2019). Banks Licensed in the Republic of Bulgaria as of 11.02.2019. [Online] Available at: http://www.bnb.bg/BankSupervision/BSCreditInstitution/BSCIRegisters/BS_CI_REG_BANKSLIST_EN [Accessed 03/03/2019].
5. Гайдаров, И. (2018). Безконтактните плащания набират популярност у нас. Computer World, 18.10.2018. [Online] Available at: https://computerworld.bg/it_liders/2018/10/18/3454903_bezkontaktnite_plashtaniia_nabirat_populiarnost_u_nas/ [Accessed 09/09/2019].
6. Димитров, Г. (2015). *Бази от данни. Практическо ръководство. Част I, Проектиране на бази от данни.* София:Издателство "Про Лангс".
7. Наков, С. и колектив. (2005). *Програмиране за .NET Framework*, том 1. ИК Фабер.
8. Петров, П. & Начева, Р. (2019). *Информационни системи за социална бизнес аналитичност в реално време*. Монографична библиотека "Проф. Цани Каляджиев", Издателство "Наука и икономика", Икономически университет - Варна, 280 стр.
9. Петров, П. & Петрова, С. (2016) Възможности за използване на протокол HTTP/2 за намаляване на лага при зареждане на уеб приложения.. *Известия на Съюза на учените - Варна, Серия "Икономически науки"* , Съюз на учените - Варна, 2016/2, с.155-165.
10. Петров, П. (2010) Сървърен софтуер, обслужващ уебсайтовете на българските банки. *Приложна информатика и статистика:*

Съвременни подходи и методи: Междунар. науч. конф., 25 - 26 септ. 2009 г., Равда, България, София: Унив. изд. Стопанство, УНСС, с.60-63.

11. Петров, П. (2013) Възможности за използване на системи от тип NoSQL за увеличаване на производителността на информационните системи. *Изв. на Съюза на учените - Варна. Сер. Икономически науки*, 2013/1, с.34-40.
12. Петров, П. (2013) Използване на уебсървърен софтуер в български и румънски банки. // *Съвременни методи и технологии в научните изследвания*: Сб. докл. от междунар. науч. конф. / Орг. к-т Тодорка Атанасова и др. - Варна: Унив. изд. Наука и икономика, 2013, с.146-152.
13. Петров, П. (2014a) Възможности за създаване на приложения, използващи WebSocket. *Научни трудове на Русенския университет*, Русенски университет, 53, серия 6.1, с.61-65.
14. Петров, П. (2014b) Еволюция в подходите за изграждане на уебприложения в реално време. // *Информационните технологии в бизнеса и образованието*: Сб. докл. от межд. науч. конф. посветена на 45 год. от създаването на кат. Информатика в ИУ - Варна. - Варна: Унив. изд. Наука и икономика, с.372-378.
15. Петров, П. (2015a) Методологични проблеми при създаването на уебприложния, работещи в реално време. *Изв. на Съюза на учените - Варна. Сер. Икономически науки, 2015/1*, с. 97-104.
16. Петров, П. (2015b) Обработка на данни във формат JSON в JavaScript - приложни подходи за работа в реално време. // *Икономиката в променящия се свят: национални, регионални и глобални измерения*: Сб. докл. от междунар. науч. конф.: Т. 3. - Варна: Унив. изд. Наука и икономика, с.155-161.
17. Петров, П. (2015c) Съвременни подходи за работа в реално време с използване на уебтехнологии. *Икономика и компютърни науки, 2015/1*, с.39-84.
18. Петров, П. (2016). Технологии, използвани в главните уеб сайтове на българските банки към средата на 2016 г. *Предизвикателствата пред информационните технологии в контекста на "Хоризонт 2020"*, Свищов: Унив. изд. Ценов, с.133-139.
19. Петров, П. (2017). Използване на протокол HTTPS в публичните сайтове на банките в България. *Известия на Съюза на учените -*

Варна Сер. Икономически науки, 2017/2, с.277-285.

20. Петров, П. (2018). Технологии за сигурност, използвани в публичните сайтове на банките в Естония. *Цифрова икономика и блокчейн технологии: Единадесета международна научноприложна конференция*, 29.06 - 01.07.2018 г. Сборник научни трудове, Варна: ЛАРГО СИТИ, 2018, с.264-271.
21. Петров, П. (2019). *Дигитализация на банковия сектор - използвани технологии в публичните уеб сайтове на банките*, Издателство "Наука и икономика", Икономически университет - Варна, 115 стр.
22. Стоянов, В. (2005). Организационна психология, Враца, ПСИДО.
23. Aas, J., Barnes, R., Case, B., Durumeric, Z., Eckersley, P., Flores-López, A., Halderman, J.A., Hoffman-Andrews, J., Kasten, J., Rescorla, E. & Schoen, S. (2019). Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. pp.2473-2487.
24. Adam, C. (2009). Considerations on Terminology and Concepts Used in the Electronic Commerce Literature. Marketing From Information to Decision, (2). pp.29-42.
25. Adi, E., Baig, Z., & Hingston, P. (2017). Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services. *Journal of Network and Computer Applications, 91*, pp.1-13.
26. Aertsen, M., Korczyński, M., Moura, G. C., Tajalizadehkhoob, S., & Van Den Berg, J. (2017). No domain left behind: is Let's Encrypt democratizing encryption?. In *Proceedings of the applied networking research workshop*. pp.48-54.
27. Alabbas, A. & Bell, J. (2018). Indexed Database API 2.0. [online] W3.org. Available at: http://www.w3.org/TR/IndexedDB/ [Accessed 19/04/2019].
28. Anand, D., & Mantrala, M. (2019). Responding to disruptive business model innovations: the case of traditional banks facing fintech entrants. *Journal of Banking and Financial Technology*, *3*(1), pp.19-31.
29. Bach, L., Mihaljevic, B., Radovan, A. (2017). Exploring HTTP/2 advantages and performance analysis using Java 9. *Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [online] pp.1522–1527. Available at: https://doi.org/10.23919/MIPRO.2017.7973663 [Accessed 05/05/2019].
30. Baker, M. (2010). *XHTML™ Basic 1.1 - Second Edition*. [online]

W3.org. Available at: http://www.w3.org/TR/xhtml-basic [Accessed 05/05/2019].
31. Banca Naţională a României (2019). REGISTRUL INSTITUTIILOR DE CREDIT LA DATA 18-01-2019, PARTEA I - Sectiunea I - Banci. [Online] Available at: http://www.bnr.ro/files/d/RegistreBNR/InstitCredit/ban1_raport.html [Accessed 03/03/2019].
32. Bank of Lithuania (2019). Supervision of financial market participants. Banks authorised in the Republic of Lithuania. [Online] Available at: https://www.lb.lt/en/sfi-financial-market-participants?business_form=82&market=1 [Accessed 07/08/2019].
33. Bank of Slovenia (2019). Institutions under supervision. [Online] Available at: https://www.bsi.si/en/financial-stability/institutions-under-supervision/banks-in-slovenia [Accessed 09/09/2019].
34. Barnes, R., Thomson, M., Pironti. A., Langley, A. (2015). Deprecating Secure Sockets Layer Version 3.0. Request for Comments (RFC) 7568. [Online] Available at: https://tools.ietf.org/rfc/rfc7568.txt [Accessed 09/09/2019].
35. Barth, A. (2011). HTTP State Management Mechanism [online] Tools.ietf.org. Available at: https://tools.ietf.org/rfc/rfc6265.txt [Accessed 05/05/2019].
36. BBC News (1999). When the saints go logging on. BBC News Sci/Tech 14.VI.1999 [Online] Available at: http://news.bbc.co.uk/2/hi/science/nature/368891.stm [Accessed 09/09/2019].
37. Belfiore, J. (2018). Microsoft Edge: Making the web better through more open source collaboration. [online] Available at: https://blogs.windows.com/windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/ [Accessed 10/01/2019].
38. Belshe, M., Peon, R. & Thomson, M. (2015). Hypertext Transfer Protocol Version 2 (HTTP/2). Request for Comments (RFC) 7540. [Online] Available at: https://tools.ietf.org/rfc/rfc7540.txt [Accessed 09/09/2019].
39. Berners-Lee, T. (1989). Information Management: A Proposal. CERN [Online] Available at: http://www.w3.org/History/1989/proposal.html
40. Berners-Lee, T., Cailliau, R. (1990). WorldWideWeb: Proposal for a

HyperText Project. [Online] Available at: http://www.w3.org/Proposal.html [Accessed 09/09/2019].

41. Berners-Lee, T., Fielding, R. & Frystyk, H. (1996). RFC 1945, Hypertext Transfer Protocol - HTTP/1.0. [online] Ietf.org Available at: http://www.ietf.org/rfc/rfc1945.txt [Accessed 05/05/2019].
42. Berners-Lee, T., Fielding, R., Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax. Request for Comments 3986. Internet Engineering Task Force. [Online] Available at: http://www.ietf.org/rfc/rfc3986.txt [Accessed 09/09/2019].
43. Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The Semantic Web. *Scientific American, May 17, 2001*. [Online] Available at: https://www.scientificamerican.com/article/the-semantic-web/ [Accessed 07/08/2019]
44. Bhogal, J. & Choksi, I. (2015). Handling Big Data Using NoSQL. *Advanced Information Networking and Applications Workshops (WAINA)*. [online] pp. 393-398. Available at: https://doi.org/10.1109/WAINA.2015.19.
45. Bhushan, K., & Gupta, B. B. (2019). Distributed denial of service (DDoS) attack mitigation in software defined network (SDN)-based cloud computing environment. *Journal of Ambient Intelligence and Humanized Computing, 10*(5), pp.1985-1997.
46. Bjørner, D. (2003). Domain Models of "The Market" - In Preparation for E-Commerce. Practical Foundations of Business and System Specifications. Kluwer.
47. Bourhis, P., Reutter, J. L., Suárez, F., & Vrgoč, D. (2017). JSON: data model, query languages and schema specification. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pp.123-135.
48. Boyer, B. (2008). Robust Java benchmarking, Part 1: Issues. [ebook] IBM Corporation. Available at: http://www.ibm.com/developerworks/java/library/j-benchmark1/j-benchmark1-pdf.pdf [Accessed 5 Oct. 2018].
49. Bozdag, E., Mesbah, A., & Van Deursen, A. (2007). A comparison of push and pull techniques for ajax. In *2007 9th IEEE International Workshop on web site evolution*, pp. 15-22.
50. Bray, T. et al. (2006). Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation 16 August 2006

(https://www.w3.org/TR/xml11/[Accessed 05/05/2019].
51. Broby, D. (2019). Strategic Fintech. Centre for Financial Regulation and Innovation: White Paper, available: https://doi.org/10.17868/68122
52. Cao, J., Wei, J., & Qin, Y. (2013). Research and Application of the Four-tier Architecture. In *International Conference of Education Technology and Information Systems (ICETIS 2013)*. https://www.atlantis-press.com/article/8022.pdf
53. Carey, H. J., & Manic, M. (2016). HTML web content extraction using paragraph tags. In *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, IEEE, pp.1099-1105.
54. Chang, X., Dou, W., Gao, Y., Wang, J., Wei, J., & Huang, T. (2019). Detecting atomicity violations for event-driven Node. js applications. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp.631-642.
55. Clark, T.& Lee, H. (1998). Security First Network Bank: a case study of an Internet pioneer. IEEE: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, vol. 4, pp.73-82.
56. Cloud.google.com. (2017). Real-time Gaming with Node.js + WebSocket on Google Cloud Platform. [online] Available at: https://cloud.google.com/solutions/real-time-gaming-with-node-js-websocket [Accessed 05/05/2019].
57. Cometd.org. (2018). V8 CometD Highly Scalable Clustered Web Messaging. [online] Available at: http://cometd.org/ [Accessed 05/05/2019].
58. Conklin, J. (1987). Hypertext: an Introduction and Survey, Computer, IX.1987, IEEE [Online] Available at: http://www.ics.uci.edu/~andre/informatics223s2007/conklin.pdf [Accessed 09/09/2019].
59. Cooper, D., Santesson, S., Farrell, S., et al. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Request for Comments (RFC) 5280. [Online] Available at: https://www.ietf.org/rfc/rfc5280.txt [Accessed 09/09/2019].
60. Country Economy (2019). Sovereigns Ratings List. [Online] Available at: https://countryeconomy.com/ratings [Accessed 09/09/2019].
61. Crockford, D. (2006). RFC 4627, The application/json Media Type for JavaScript Object Notation (JSON). [online] Available at: http://www.ietf.org/rfc/rfc4627.txt [Accessed 05/05/2019].

62. Czech National Bank (2019). Banks and branches of foreign banks. [Online] Available at: https://apl.cnb.cz/apljerrsdad/JERRS.WEB15.BASIC_LISTINGS_RESPONSE_3[Accessed 09/09/2019].
63. Davis, J. C., Williamson, E. R., & Lee, D. (2018). A sense of time for javascript and node. js: First-class timeouts as a cure for event handler poisoning. In *27th USENIX Security Symposium*, pp.343-359.
64. DB-Engines. (2017a). DB-Engines Ranking. [online] Available at: https://db-engines.com/en/ranking/key-value+store [Accessed 05/05/2019].
65. DB-Engines. (2017b). DB-Engines Ranking. [online] Available at: http://db-engines.com/en/ranking/relational+dbms [Accessed 05/05/2019].
66. De Bono, E., & Zimbalist, E. (1970). *Lateral thinking*. London: Penguin Books.
67. Developer.apple.com. (2019). Apple Filing Protocol Programming Guide. [online] Available at: https://developer.apple.com/library/content/documentation/Networking/Conceptual/AFP/ [Accessed 19/04/2019].
68. Developer.mozilla.org. (2018). Creating JavaScript callbacks in components. [online] Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Creating_JavaScript_callbacks_in_components#JavaScript_functions_as_callbacks [Accessed 05/05/2019].
69. Developers.google.com. (2018). *V8 JavaScript engine*. [online] Available at: https://developers.google.com/v8/ [Accessed 05/05/2019].
70. Dierks, T. & Allen, C. (1999). The TLS Protocol Version 1.0. Request for Comments (RFC) 2246. [Online] Available at: https://www.rfc-editor.org/rfc/rfc2246.txt [Accessed 09/09/2019].
71. Dierks, T. & Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. Request for Comments (RFC) 4346. [Online] Available at: https://tools.ietf.org/rfc/rfc4346.txt [Accessed 09/09/2019].
72. Dierks, T. & Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. Request for Comments (RFC) 5246. [Online] Available at: https://tools.ietf.org/rfc/rfc5246.txt [Accessed 09/09/2019].
73. Drasch, B. J., Schweizer, A., & Urbach, N. (2018). Integrating the 'Troublemakers': A taxonomy for cooperation between banks and fintechs. *Journal of Economics and Business, 100*, pp.26-42.

https://doi.org/10.1016/j.jeconbus.2018.04.002
74. Earnshaw, R., Guedj, R., Dam, A. & Vince, J. (2012). *Frontiers of Human-Centered Computing, Online Communities and Virtual Environments*. Springer Science & Business Media.
75. Ecma International. (2011). Standard ECMA-262 5.1 Edition, ECMAScript Language Specification. [ebook] ECMA International. Available at: http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf [Accessed 05/05/2019].
76. Ecma International. (2017). Standard ECMA-404, The JSON Data Interchange Syntax. 2nd ed. [ebook] ECMA International. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf [Accessed 05/05/2019].
77. Eco, U. (1996). From Internet to Gutenberg. [Online] Available at: http://www.hf.ntnu.no/anv/Finnbo/tekster/Eco/Internet.htm [Accessed 09/09/2019].
78. El-Hassan, F., Peterkin, R., Abou-Gabal, M., & Ionescu, D. (2009). A high-performance architecture of an XML processor for SIP-based presence. In *2009 Sixth International Conference on Information Technology: New Generations*, IEEE, pp.90-95.
79. Elshaiekh, N. E. M., Hassan, Y. A. A., & Abdallah, A. A. A. (2018). The Impacts of Remote Working on Workers Performance. In *2018 International Arab Conference on Information Technology (ACIT)*, IEEE, pp.1-5. doi: 10.1109/ACIT.2018.8672704
80. Estonia Finantsinspektsioon (2018). Licensed Credit Institutions in Estonia, Supervised Entities. [Online] Available at: https://www.fi.ee/index.php?id=3163 [Accessed 10/04/2018]
81. Estonia Finantsinspektsioon (2019). Supervised Entities. Licensed credit institutions in Estonia [Online] Available at: https://www.fi.ee/en/banking-and-credit/banking-and-credit/credit-institutions/licensed-credit-institutions-estonia [Accessed 07/08/2019].
82. Eurostat (2019), Gross domestic product at market prices. Eurostat Database [Online] Available at: https://ec.europa.eu/eurostat/tgm/table.do?tab=table&plugin=1&language=en&pcode=tec00001 [Accessed 07/08/2019].
83. Eurostat (2019). Gini coefficient of equivalised disposable income - EU-SILC survey. [Online] Available at: https://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&language=e

n&pcode=tessi190&plugin=1 [Accessed 09/09/2019].

84. Fette, I. & Melnikov, A. (2011). RFC 6455, The WebSocket Protocol. [online] Tools.ietf.org. Available at: http://tools.ietf.org/rfc/rfc6455.txt [Accessed 05/05/2019].
85. Firebird Foundation Inc. (2019). Firebird: The true open source database for Windows, Linux, Mac OS X and more. [online] Available at: http://www.firebirdsql.org/ [Accessed 19/04/2019].
86. Freier, A., Karlton, P. & Kocher, P. (2011). The Secure Sockets Layer (SSL) Protocol Version 3.0. Request for Comments (RFC) 6106. [Online] Available at: https://tools.ietf.org/rfc/rfc6101.txt [Accessed 09/09/2019].
87. Friedman, B. G. (2004). *Web search savvy: Strategies and shortcuts for online research*. Psychology Press.
88. Goel, U., Steiner, M., Wittie, M. P., Ludin, S., & Flack, M. (2017). Domain-Sharding for faster HTTP/2 in lossy cellular networks. arXiv preprint. https://arxiv.org/pdf/1707.05836.pdf
89. González, A. (2003). PayPal and eBay: The legal implications of the C2C electronic commerce model. *18th BILETA Conference: Controlling Information in the Online Environment*, QMW London. [Online] Available at: https://pdfs.semanticscholar.org/9b4d/7c82e9e525ba27a05346c034eb3154b40708.pdf [Accessed 09/09/2019].
90. Groves, K. S., & Vance, C. M. (2015). Linear and nonlinear thinking: A multidimensional model and measure. *The Journal of Creative Behavior, 49*(2), pp.111-136.
91. Gupta, A., (2017). Create a WebSocket application using Oracle Developer Cloud & Application Container Cloud. [online] Available at: https://community.oracle.com/community/cloud_computing/oracle-cloud-developer-solutions/blog/2017/02/02/building-a-websocket-application-using-oracle-developer-cloud-application-container-cloud [Accessed 09/09/2019].
92. Hadoop.apache.org. (2019). Apache Hadoop. [online] Available at: http://hadoop.apache.org/ [Accessed 19/04/2019].
93. Haour, G. (2019). The crucial human factor in innovation. On Research - Journal of EU Business School, pp.76-99. https://onresearch.ch/wp-content/uploads/2019/05/ONRESEARCH-research-journal-ed2-v2-soft.pdf

94. Hernández, E., González, A., Pérez, B., de Luis Reboredo, A., & Rodríguez, S. (2018). Virtual organization for fintech management. In *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, pp.201-210.
95. Hickman, K. (1995). The SSL Protocol. Internet Draft. [Online] Available at: https://tools.ietf.org/id/draft-hickman-netscape-ssl-00.txt [Accessed 09/09/2019].
96. Hickson, I. (2010). Web SQL Database, W3C Working Group Note. [online] W3.org. Available at: https://www.w3.org/TR/webdatabase/ [Accessed 05/05/2019].
97. Hickson, I. (2012). The WebSocket API, W3C Candidate Recommendation. [online] W3.org. Available at: (http://www.w3.org/TR/websockets/ [Accessed 05/05/2019].
98. Hickson, I. (2015). Server-Sent Events, W3C Recommendation. [online] W3.org. Available at: https://www.w3.org/TR/eventsource/ [Accessed 05/05/2019].
99. Hickson, I. (2016). Web Storage (Second Edition). W3C Recommendation. [online] W3.org. Available at: https://www.w3.org/TR/webstorage/ [Accessed 05/05/2019].
100. Hispalensis, I. (circa 600). Isidori Hispalensis Episcopi Etymologiarum sive Originum libri XX, ed. W. M. Lindsay, Oxford 1911. [Online] Available at: http://www.hs-augsburg.de/~harsch/Chronologia/Lspost07/Isidorus/isi_et00.html [Accessed 09/09/2019].
101. Hodges, J., Jackson, C. & Barth, A., (2012). HTTP Strict Transport Security (HSTS), IETF RFC6797
102. Httparchive.org. (2019). State of the Web. [online] Available at: http://httparchive.org/trends.php?s=All [Accessed 19/04/2019].
103. Httpd.apache.org. (2018a). Environment Variables in Apache - Apache HTTP Server Version 2.4. [online] Available at: http://httpd.apache.org/docs/2.4/env.html#special [Accessed 05/05/2019].
104. Httpd.apache.org. (2018b). HTTP/2 guide - Apache HTTP Server Version 2.4. [online] Available at: https://httpd.apache.org/docs/2.4/howto/http2.html [Accessed 05/05/2019].
105. IETF. (2018). Memos in the Requests for Comments (RFC) document

series contain technical and organizational notes about the Internet. [online] Available at: https://www.ietf.org/standards/rfcs/ [Accessed 05/05/2019].

106. IETF. (2019). Internet Engineering Task Force. [online] Available at: https://www.ietf.org/ [Accessed 19/04/2019].
107. Ilmudeen, A., Bao, Y., & Alharbi, I. M. (2019). How does business - IT strategic alignment dimension impact on organizational performance measures: Conjecture and empirical analysis. *Journal of Enterprise Information Management, 32*(3), pp.457-476. https://doi.org/10.1108/JEIM-09-2018-0197, https://drive.google.com/file/d/18FB8b4Ix9WEC77To2LYtBnh_0kiY6Fu3/view
108. Islam, N. S., Lu, X., Wasi-ur-Rahman, M., Shankar, D., & Panda, D. K. (2015). Triple-H: A hybrid approach to accelerate HDFS on HPC clusters with heterogeneous storage architecture. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.101-110.
109. ISO/IEC 15445:2000(E), Information technology. Document description and processing languages. HyperText Markup Language (HTML) http://purl.org/NET/ISO+IEC.15445/15445.html
110. Json.org. (2019). Introducing JSON. [online] Available at: http://www.json.org/ [Accessed 19/04/2019].
111. Karakas, C., & Stamegna, C. (2018). Defining an EU-Framework for Financial Technology (FinTech): Economic Perspectives and Regulatory Challenges. *Law and Economics Yearly Review, 7*(1), pp.106-29. https://www.laweconomicsyearlyreview.org.uk/Law_and_Economics_Yearly_Review_LEYR_Journal_vol_7_part_1_2018.pdf
112. Kattel, R. & Mergel, I. (2018). Estonia's digital transformation: Mission mystique and the hiding hand. UCL Institute for Innovation and Public Purpose working Paper Series (IIPP WP 2018-09).
113. Kesteren, A., Aubourg, J., Song, J. & Steen, H. (2016). XMLHttpRequest *Level 1*. [online] W3.org. Available at: https://www.w3.org/TR/XMLHttpRequest/ [Accessed 5 Jan. 2016].
114. Kitsios, F., & Kamariotou, M. (2019). Strategizing Information Systems: An Empirical Analysis of IT Alignment and Success in SMEs. *Computers, 8*(4):74. https://doi.org/10.3390/computers8040074
115. Koltsidas, I., & Viglas, S. D. (2011). Designing a flash-aware two-level

cache. In *East European Conference on Advances in Databases and Information Systems*, Springer, pp.153-169.

116. Krawiec, P., Sosnowski, M., Batalla, J. M., Mavromoustakis, C. X., Mastorakis, G., & Pallis, E. (2017). Survey on technologies for enabling real-time communication in the web of things. In *Beyond the Internet of Things*, Springer, pp.323-339.
117. Kula, R. G., Ouni, A., German, D. M., & Inoue, K. (2017). On the impact of micro-packages: An empirical study of the npm javascript ecosystem. arXiv preprint arXiv:1709.04638.
118. Kumar, B., Abhishek, K., Deepak, A., & Singh, M. P. (2016). Implementation of interactive real time online co-shopping using Push AJAX. Procedia Computer Science, 89, pp.473-482.
119. Latvia Financial And Capital Market Commission (2019). Market. Credit institutions. Banks. [Online] Available at: https://www.fktk.lv/en/market/credit-institutions/banks/ [Accessed 07/08/2019].
120. Loreto, S., P. Saint-Andre, S. Salsano, G. Wilkins. (2011). *RFC 6202*, Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. [online] Available at: http://www.ietf.org/rfc/rfc6202.txt [Accessed 19/04/2019].
121. Loring, M. C., Marron, M., & Leijen, D. (2017). Semantics of asynchronous JavaScript. In *Proceedings of the 13th ACM SIGPLAN International Symposium on on Dynamic Languages*, pp. 51-62.
122. Lv, T., Yan, P., & He, W. (2018). Survey on JSON data modelling. In *Journal of Physics: Conference Series*, *1069*(1), p.012101.
123. Maata, R. L. R., Cordova, R., Sudramurthy, B., & Halibas, A. (2017). Design and implementation of Client-Server based application using socket programming in a distributed computing environment. In *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp.1-4.
124. Maeda, K. (2012). Performance evaluation of object serialization libraries in XML, JSON and binary formats. *Digital Information and Communication Technology and it's Applications (DICTAP)*. [online] pp. 177 – 182, Available at: https://doi.org/10.1109/DICTAP.2012.6215346 [Accessed 05/05/2019].
125. Magyar Nemzeti Bank (2019). The register of the financial service providers. [Online] Available at:

https://intezmenykereso.mnb.hu/en/Home/Index [Accessed 09/09/2019].
126. Maignant, C. (2017). The Irish Catholic Church and the Internet. *New Hibernia Review, 21*(4), pp.20-38.
127. Mardan, A. (2018). Real-Time Apps with WebSocket, Socket. IO, and DerbyJS. In *Practical Node.js*. Apress, Berkeley, CA., pp.307-330.
128. Marszałkowski, J., Mizgajski, J., Mokwa, D., & Drozdowski, M. (2015). Analysis and solution of CSS-sprite packing problem. *ACM Transactions on the Web (TWEB), 10*(1), pp.1-34.
129. Martin-Flatin, J. P. (1999). Push vs. pull in Web-based network management. *Integrated Network Management*. [online] pp. 3 - 18, Available at: https://doi.org/10.1109/INM.1999.770671 [Accessed 05/05/2019].
130. MDN Web Docs. (2018). DOMParser. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/DOMParser [Accessed 05/05/2019].
131. Megaw, N. & Crow, D. (2018). Customers' money trapped at UK fintech due to Malta bank freeze. *Financial Times 25.10.2018*. [Online] Available at: https://www.ft.com/content/64caa038-d79a-11e8-a854-33d6f82e62f8 [Accessed 09/09/2019].
132. Micro Focus (2018). *Open Enterprise Server*. [online] Available at: https://www.microfocus.com/products/open-enterprise-server/ [Accessed 05/05/2019].
133. Miranda, M. (2006). Hipertexto e Medievalidade. Enciclopédia e Hipertexto. [Online] Available at: http://www.educ.fc.ul.pt/hyper/resources/amiranda/index.htm [Accessed 09/09/2019].
134. Mjelde, E., & Opdahl, A. L. (2017). Load-time reduction techniques for device-agnostic web sites. *Journal of Web Engineering*, pp.311-346.
135. Munns, C. (2018). Announcing WebSocket APIs in Amazon API Gateway. AWS Compute Blog. [online] Available at: https://aws.amazon.com/blogs/compute/announcing-websocket-apis-in-amazon-api-gateway/ [Accessed 09/09/2019].
136. Mysql.com. (2018). MySQL Workbench. [online] Available at: https://www.mysql.com/products/workbench/ [Accessed 05/05/2019].
137. National Bank of Serbia (2019), List of Banks as of 22.03.2019. [Online] Available at: https://www.nbs.rs/internet/english/ 50/50_2.html [Accessed 03/03/2019].

138. National Institute of Statistics (2018). Population. [Online] Available at: http://www.insse.ro/cms/sites/default/files/com_presa/com_pdf/poprez_ian2018e.pdf [Accessed 03/03/2019].
139. National Statistical Institute (2018). Population by districts, municipalities, place of residence and sex as of 31.12.2018. [Online] Available at: http://www.nsi.bg/en/content/6704/ population-districts-municipalities-place-residence-and-sex [Accessed 03/03/2019].
140. Nelson, T. (1965). Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference (ACM '65)*. ACM, New York, NY, USA, pp.84-100.
141. Nodejs.org. (2018). Node.js [online] Available at:http://www.nodejs.org/ [Accessed 05/05/2019].
142. Novozhilov, D., Kotenko, I., & Chechulin, A. (2016). Improving the categorization of web sites by analysis of html-tags statistics to block inappropriate content. In *Intelligent Distributed Computing IX*, Springer, pp.257-263
143. Npmjs.com. (2015). BPMN. [online] Available at: https://www.npmjs.com/package/bpmn [Accessed 10/01/2019].
144. Npmjs.com. (2018a). Node Package Manager. [online] Available at: https://www.npmjs.com [Accessed 05/05/2019].
145. Npmjs.com. (2018b). npm-package.json | Specifics of npm's package.json handling. [online] Available at: https://docs.npmjs.com/files/package.json [Accessed 5 Apr. 2019].
146. Npmjs.org. (2017a). A W3C Standard XML DOM(Level2 CORE) implementation and parser (DOMParser/XMLSerializer). [online] Available at: https://www.npmjs.org/package/xmldom [Accessed 05/05/2019].
147. Npmjs.org. (2017b). An evented streaming XML parser in JavaScript. [online] Available at: https://www.npmjs.org/package/sax. [Accessed 05/05/2019].
148. Npmjs.org. (2017c). Fast & forgiving HTML/XML/RSS parser. [online] Available at: https://www.npmjs.org/package/htmlparser2 [Accessed 05/05/2019].
149. Npmjs.org. (2017d). Simple XML to JavaScript object converter. [online] Available at: https://www.npmjs.org/package/xml2js [Accessed 05/05/2019].

150. Npmjs.org. (2018a). libxml bindings for v8 javascript engine. [online] Available at: https://www.npmjs.org/package/libxmljs [Accessed 19/11/2018].
151. Npmjs.org. (2019). NPM. [online] Available at: https://www.npmjs.org/ [Accessed 19/04/2019].
152. Openafs.org. (2019). OpenAFS. [online] Available at: http://www.openafs.org [Accessed 19/04/2019].
153. Oracle. (2019). Oracle Berkeley DB. [online] Available at: http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html [Accessed 19/04/2019].
154. Panayotova, G., Dimitrov, G., Petrov, P. & Bychkov, O. (2016). Modeling and data processing of information systems. *3rd International Conference on Artificial Intelligence and Pattern Recognition (AIPR), Lodz, Poland, IEEE*, pp. 154-158.
155. Pandey, M., & Pandey, R. (2017). JSON and its use in Semantic Web. *International Journal of Computer Applications, 164*(11), pp.10-16.
156. Peon, R. & Ruellan, H. (2015). HPACK: Header Compression for HTTP/2. [online] Tools.ietf.org. Available at: https://tools.ietf.org/rfc/rfc7541.txt [Accessed 05/05/2019].
157. Petrov, P. & Dimitrov, P. (2019). Security Certificates in Public Web Sites of Banks from Balkan States. *9th International Conference On Application Of Information And Communication Technology And Statistics In Economy And Education (ICAICTSEE - 2019)*, UNWE, Sofia, Bulgaria, pp.160-169.
158. Petrov, P. & Hundal, S. (2018). Application of Security Technologies in the Public Websites of Banks in Serbia. *Izvestia Journal of the Union of Scientists - Varna. Economic Sciences Series*, Varna: Union of Scientists - Varna, 7(2), pp.298-305.
159. Petrov, P. & Valov, N. (2019a). Digitalization of Banking Services and Methodology for Building and Functioning of Fintech Companies. *Izvestia Journal of the Union of Scientists - Varna. Economic Sciences Series*, Varna : Union of Scientists - Varna, 8(1), pp.110-117.
160. Petrov, P. & Valov, N. (2019b). Strategic and Tactical Problems in Fintech and E-business Companies. *Izvestia Journal of the Union of Scientists - Varna. Economic Sciences Series*, Varna : Union of Scientists - Varna, 8(3), pp.55-61.
161. Petrov, P. (2013) Trends in The Use of Web Server Software in

Bulgarian Banks. *International Conference on Application of Information and Communication Technology and Statistics In Economy And Education (ICAICTSEE-2012)* Conference Proceedings, UNWE, c.359-364.

162. Petrov, P. (2019). Web security technologies used in banks of Estonia, Latvia and Lithuania. *Information and Communication Technologies in Business and Education*: Proceedings of the International Conference Dedicated to the 50th Anniversary of the Department of Informatics, Varna: Science and Economic Publ. House, pp.145-154.
163. Petrov, P., Buevich, A., Dimitrov, G., Kostadinova, I. & Dimitrov, P. (2019a). A Comparative Study on Web Security Technologies Used in Bulgarian and Serbian Banks. *19 International Multidisciplinary Scientific Geoconference SGEM 2019*: Conference Proceedings, 28 June - 7 July 2019, Albena, Bulgaria: Vol. 19. Informatics, Iss. 2.1, Sofia: STEF92 Technology Ltd., pp.3-10. SCOPUS
164. Petrov, P., Dimitrov, G. & Ivanov, S. (2018a). A Comparative Study on Web Security Technologies Used in Irish and Finnish Banks. *18 International Multidisciplinary Scientific Geoconference SGEM 2018*: Conference Proceedings, 2 - 8 July 2018, Albena, Bulgaria: Vol. 18. Informatics, Geoinformatics a. Remote Sensing. Iss. 2.1. Informatics, Sofia: STEF92 Technology Ltd., Vol. 18, 2018, Iss. 2.1, pp.3-10. SCOPUS
165. Petrov, P., Krumovich, S., Nikolov, N., Dimitrov, G. & Sulov, V. (2018b). Web Technologies Used in the Commercial Banks in Finland. CompSysTech'18: *19-th International Conference on Computer Systems and Technologies*, 13-14 September 2018, University of Ruse, Bulgaria, New York, USA : ACM [Association for Computing Machinery], 2018, pp.94-98. SCOPUS
166. Petrov, P., Malkawi, R., Shichkin, A., Dimitrov, G. & Nacheva, R. (2019b) Security Certificates Used in Public Web Sites of Banks in Czech Republic, Slovakia and Hungary. *TEM Journal, 8*(4), pp.1224-1231. SCOPUS
167. Petrov, P., Sulov, V., Parusheva, S., Penchev, B. & Collins, J. (2017). Web Technologies Used in the Home Pages of the Irish Banks. *4th International Multidisciplinary Scientific Conference on Social Sciences & Arts SGEM 2017*, Sofia: STEF92 Technology Ltd., 5, 2017, pp.1135-1142.

168. Philip, G. (1998). Software design guidelines for event-driven programming. *Journal of Systems and Software*. 41(2). [online] pp. 79-91 Available at: https://doi.org/10.1016/S0164-1212(97)10009-7 [Accessed 05/05/2019].
169. Pixley, T. (1999). Document Object Model Events. [online] Available at: https://www.w3.org/TR/WD-DOM-Level-2/events.html [Accessed 05/05/2019].
170. Postel, J. (1981). Transmission Control Protocol. Request for Comments (RFC) 793. [Online] Available at: https://tools.ietf.org/rfc/rfc793.txt [Accessed 09/09/2019].
171. Prajitno, H. W., Kristanda, M. B., & Kusnadi, A. (2019). Study of Push and Pull Techniques in Delivering Mobile Application Notifications. In *2019 5th International Conference on New Media Studies (CONMEDIA)*, IEEE, pp.20-25.
172. Puranik, D., Feiock, D. & Hill, J. (2013). Real-Time Monitoring using AJAX and WebSockets. *Engineering of Computer Based Systems (ECBS)*. [online] pp. 110-118 Available at: https://doi.org/10.1109/ECBS.2013.10 [Accessed 05/05/2019].
173. Qualys SSL Labs. (2017). SSL and TLS Deployment Best Practices. [Online] Available at: https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices [Accessed 09/09/2019].
174. Redis.io. (2019). Redis. [online] Available at: https://redis.io [Accessed 5 Jan. 2019].
175. Rehman, A. U., & Anwar, M. (2019). Mediating role of enterprise risk management practices between business strategy and SME performance. *Small Enterprise Research*, *26*(2), pp.207-227.
176. Rescorla, E. (2000). HTTP Over TLS. Request for Comments (RFC) 2818. [Online] Available at: https://tools.ietf.org/rfc/rfc2818.txt [Accessed 09/09/2019].
177. Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. Request for Comments (RFC) 8446. [Online] Available at: https://tools.ietf.org/rfc/rfc8446.txt [Accessed 09/09/2019].
178. Richards, G., Hammer, C., Burg, B., & Vitek, J. (2011). The eval that men do. In *European Conference on Object-Oriented Programming*. Springer, pp.52-78.
179. Robinson, M. (2017). How to Get Free HTTPS Certificates from Let's Encrypt. *Journal of Intellectual Freedom & Privacy*, *2*(1), pp.11-12.

180. Romānova, I. & Kudinska, M. (2016). Banking and Fintech: A Challenge or Opportunity? Contemporary Issues in Finance: Current Challenges from Across Europe (*Contemporary Studies in Economic and Financial Analysis, Vol. 98*), Emerald Group Publishing Limited, pp.21-35.
181. Roonemaa, M. (2017). Global lessons from Estonia's tech-savvy government. *The UNESCO Courier. April-June*, pp.2220-2293.
182. Ross, D. & Gondrom, T. (2013). HTTP Header Field X-Frame-Options. Request for Comments (RFC) 7034. [Online] Available at: https://tools.ietf.org/rfc/rfc7034.txt [Accessed 09/09/2019].
183. Ross, D., Gondrom, T. & Stanley,. T., (2013). HTTP Header Field X-Frame-Options, IETF RFC7034.
184. Saint-Andre, P. & Hodges, J. (2011). Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS). RFC6125. [Online] Available at: https://tools.ietf.org/rfc/rfc6125.txt [Accessed 07/08/2019].
185. Saint-Andre, P., Hodges, J. (2011). Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS). Request for Comments (RFC) 6125. [Online] Available at: https://tools.ietf.org/rfc/rfc6125.txt [Accessed 09/09/2019].
186. Sajić, M., Bundalo, D., Bundalo, Z., & Pašalić, D. (2017). Digital technologies in transformation of classical retail bank into digital bank. In *2017 25th Telecommunication Forum (TELFOR)*, IEEE, pp.1-4.
187. Samba.org. (2019). *SAMBA.* [online] Available at: https://www.samba.org/ [Accessed 10/01/2019].
188. Santos, A. L., Valente, M. T., & Figueiredo, E. (2015). Using JavaScript static checkers on GitHub systems: A first evaluation. In *Proccedings of the 3rd workshop on software visualization, evolution and maintenance (VEM)*, pp.33-40.
189. Sarhan, A. M., Hamissa, G. M., & Elbehiry, H. E. (2015). Feature Selection algorithms based on HTML tags importance. In *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*, IEEE, pp.185-190.
190. Schackow, S. (2013). Introduction to WebSockets on Windows Azure

Web Sites. [online] Available at: https://azure.microsoft.com/en-us/blog/introduction-to-websockets-on-windows-azure-web-sites/ [Accessed 09/09/2019].
191. Schueffel, P. (2017). Taming the Beast: A Scientific Definition of Fintech. Journal of Innovation Management. 4 (4). pp.32–54
192. Semver.org. (2019). *Semantic Versioning 2.0.0*. [online] Available at: http://semver.org/ [Accessed 10/01/2019].
193. Seville, I. (circa 600). Isidore of Seville: The Etymologies (or Origins). [Online] Available at: http://penelope.uchicago.edu/Thayer/E/Roman/Texts/Isidore/home.html [Accessed 09/09/2019].
194. Sharon, A., & Dori, D. (2017). Model-Based Project-Product Lifecycle Management and Gantt Chart Models: A Comparative Study. *Systems engineering, 20*(5), pp.447-466.
195. Shepler, S. et al. (2010). *RFC 5661, Network File System (NFS) Version 4 Minor Version 1 Protocol*. [online] Available at: http://tools.ietf.org/rfc/rfc5661.txt [Accessed 05/05/2019].
196. Skvarciany, V. & Jurevičienė, D. (2017). Factors affecting personal customers' trust in traditional banking: case of the Baltics, *Journal of Business Economics and Management, 18*(4), pp. 636-649.
197. Socket.io. (2017). *Socket.IO 2.0.1, 2.0.2 and 2.0.3*. [online] Available at: http://socket.io/blog [Accessed 19/04/2019].
198. Socket.IO. (2019). *Socket.IO: Featuring the fastest and most reliable real-time engine*. [online] Available at: https://socket.io [Accessed 19/04/2019].
199. Sqlite.org. (2017). *SQLite Database Speed Comparison*. [online] Available at: https://sqlite.org/speed.html [Accessed 05/05/2019].
200. Sqlite.org. (2018). *SQL As Understood By SQLite*. [online] Available at: http://www.sqlite.org/lang.html [Accessed 05/05/2019].
201. Sqlite.org. (2019). *SQLite*. [online] Available at: http://www.sqlite.org [Accessed 19/04/2019].
202. sqlite3 API. (2018). [online] Available at: https://github.com/mapbox/node-sqlite3/wiki/API [Accessed 05/05/2019].
203. Statcounter.com (2019). Desktop Browser Market Share Worldwide Jan 2009 - Sept 2019. [Online] Available at: https://gs.statcounter.com/browser-market-

share/desktop/worldwide/#monthly-200901-201909 [Accessed 09/09/2019].
204. Statistical Office of the Republic of Serbia (2017). Estimated population. [Online] Available at: http://www.stat.gov.rs/en-us/vesti/20180629-procene-stanovnistva-2017/?s=1801 [Accessed 03/03/2019].
205. Sulov, V. (2014). On the Essence of Hardware Performance. *Research Journal of Economics, Business and ICT*, 9(1), pp.13-18.
206. Suri, N., Fronteddu, R., Cramer, E., Breedy, M., Marcus, K., in't Velt, R., Nilsson, J., Mantovani, M., Campioni, L., Poltronieri, F., & Benincasa, G. (2018). Experimental evaluation of group communications protocols for tactical data dissemination. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, IEEE, pp.133-139.
207. Suzumura, T. & Oiki, T. (2011). StreamWeb: Real-Time Web Monitoring with Stream Computing. *Washington: IEEE International Conference on Web Services*, pp. 620-627.
208. Tammpuu, P. & Masso, A. (2018). 'Welcome to the virtual state': Estonian e-residency and the digitalised state as a commodity. *European Journal of Cultural Studies, 21*(5), pp.543-560.
209. Tilkov, S., & Vinoski, S. (2010). Node. js: Using JavaScript to build high-performance network programs. IEEE *Internet Computing, 14*(6), pp.80-83.
210. Titu, M. A., Stanciu, A., & Titu, S. (2018). Business Process Outsourcing. Integrity in an era of digital transformation. *Journal of Electrical Engineering, Electronics, Control and Computer Science, 4*(1), pp.1-4.
211. Tripathi, N., & Hubballi, N. (2018). Slow rate denial of service attacks against HTTP/2 and detection. *Computers & security, 72*, pp.255-272.
212. Turner, S. & Polk, T. (2011). Prohibiting Secure Sockets Layer (SSL) Version 2.0. Request for Comments (RFC) 6176. [Online] Available at: https://tools.ietf.org/rfc/rfc6176.txt [Accessed 09/09/2019].
213. UN Human Development Report Office (2019), Human Development Indices and Indicators. 2018 Statistical update. [Online] Available at: http://hdr.undp.org/en/2018-update [Accessed 07/08/2019].
214. United Nations Development Programme (2019). Human Development Index and its components. [Online] Available at: http://hdr.undp.org/en/composite/HDI [Accessed 09/09/2019].
215. Van Der Hooft, J., Petrangeli, S., Wauters, T., Huysegems, R., Bostoen,

T., & De Turck, F. (2018). An HTTP/2 push-based approach for low-latency live streaming with super-short segments. *Journal of Network and Systems Management, 26*(1), pp.51-78.

216. Vance, C. M., Groves, K. S., Paik, Y., & Kindler, H. (2007). Understanding and measuring linear–nonlinear thinking style for enhanced management education and professional practice. *Academy of Management Learning & Education, 6*(2), pp.167-185.
217. Vijaya, A., & Venkataraman, N. (2018). Modernizing legacy systems: A re-engineering approach. *International Journal of Web Portals (IJWP), 10*(2), pp.50-60.
218. Walker, R. (2006). *Examining Load Average, Linux Journal*. [Online] Available at: https://www.linuxjournal.com/article/9001 [Accessed 4 Feb. 2019].
219. Wallace, P. (2001). *Encyclopedia of E-commerce/vol.III*. New Delhi, Sarup & Sons, p.1040, [Online] Available at: http://books.google.com/books?id=-cbu43p7gfUC [Accessed 09/09/2019].
220. Wang, K. C. (2018). TCP/IP and Network Programming. In *Systems Programming in Unix/Linux*. Springer, pp.377-412.
221. Wenzel, M., & Meinel, C. (2015). Parallel network data processing in client side JavaScript applications. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, IEEE, pp.140-147.
222. Wijnants, M., Marx, R., Quax, P., & Lamotte, W. (2018). HTTP/2 prioritization and its impact on web performance. In *Proceedings of the 2018 World Wide Web Conference*, pp. 1755-1764.
223. Wikipedia (2019). Baltic states. Wikipedia.org. [Online] Available at: https://en.wikipedia.org/wiki/Baltic_states [Accessed 07/08/2019].
224. Wittern, E., Suter, P., & Rajagopalan, S. (2016). A look at the dynamics of the JavaScript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories,* pp.351-361.
225. World Bank (2019). National accounts data and OECD National Accounts data files. [Online] Available at: https://data.worldbank.org/indicator/NY.GNP.PCAP.CD [Accessed 09/09/2019].
226. World Bank Development Research Group (2019), GINI index (World Bank estimate). International Comparison Program database. [Online] Available at:

https://data.worldbank.org/indicator/NY.GNP.PCAP.PP.CD [Accessed 07/08/2019]

227. Yamamoto, K., Tsujikawa, T., & Oku, K. (2017). Exploring HTTP/2 Header Compression. In *Proceedings of the 12th International Conference on Future Internet Technologies*, pp.1-5.
228. Yao, Q., Sun, P., Hu, L., Zhu, X., Ni, H. (2008). Effective Iframe-Based Strategy for Processing Dynamic Data in Embedded Browser. *Advanced Computer Theory and Engineering*. [online] pp. 538-542 Available at: https://doi.org/10.1109/ICACTE.2008.23 [Accessed 05/05/2019].
229. Yue, C., & Wang, H. (2013). A measurement study of insecure javascript practices on the web. *ACM Transactions on the Web (TWEB), 7*(2), pp.1-39.
230. Zafar, R., Yafi, E., Zuhairi, M., Dao, H. (2016). Big Data: The NoSQL and RDBMS review. *Information and Communication Technology (ICICTM)*, pp.120–126. [online] Available at: https://doi.org/10.1109/ICICTM.2016.7890788 [Accessed 05/05/2019].
231. Zapata, R. E., Kula, R. G., Chinthanet, B., Ishio, T., Matsumoto, K., & Ihara, A. (2018). Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm javascript packages. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp.559-563.
232. Zarifis, K., Holland, M., Jain, M., Katz-Bassett, E., & Govindan, R. (2016). Modeling HTTP/2 speed from HTTP/1 traces. In *International Conference on Passive and Active Network Measurement*. Springer, pp.233-247.
233. Zavolokina, L., Dolata, M., & Schwabe, G. (2016). FinTech transformation: How IT-enabled innovations shape the financial sector. In *FinanceCom 2016*. Springer, pp.75-88.
234. Zekri, M., El Kafhali, S., Aboutabit, N., & Saadi, Y. (2017). DDoS attack detection using machine learning techniques in cloud computing environments. In *2017 3rd international conference of cloud computing technologies and applications (CloudTech)*, IEEE, pp.1-7.
235. Zimmermann, T., Rüth, J., Wolters, B., & Hohlfeld, O. (2017). How HTTP/2 pushes the web: An empirical study of HTTP/2 server push. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, IEEE, pp.1-9.

ПРИЛОЖЕНИЯ

Приложение 1. Пример за използване на стратегия "Издърпване"

Използва се вътрешен фрейм iframe за постоянно извеждане на точното време при уеб сървъра.

Файл time_pull.js:

```
var http = require('http');
http.createServer(function (req, res) {
	res.writeHead(200, {'Content-Type': 'text/html'});
	if(req.url == '/' || req.url == '/index.html') {
		res.end('<html> \
			<head><title>Server Clock</title></head> \
			<body> \
				<div id="time">Clock</div> \
				<iframe src="frame.html" id="ifr"> \
				</iframe> \
				<script> \
					setInterval(function(){ document.getElementById("ifr").src
= document.getElementById("ifr").src;}, 10000); \
				</script> \
			</body> \
		</html> \
		');
		console.log('200: ' + req.url);
	} else if(req.url == '/frame.html') {
		d = new Date();
		res.end('<html> \
			<head><meta http-equiv="refresh" content="3" /></head> \
			<body> \
				<script> \
```

```
                        parent.document.getElementById("time").innerHTML = "' +
d.toString() + "'; \
                    </script> \
                </body> \
            </html> \
            ');
            console.log('200: ' + req.url);
        } else {
            res.end('<H1>ERROR 404</H1>');
            console.log('err404: ' + req.url);
        }
}).listen(1234, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1234/');
```

В програмния код по-горе, родителският фрейм презарежда вътрешния фрейм през 10 сек. (10000 мс), а вътрешният фрейм презарежда сам себе си през 3 сек. При това положение вътрешният фрейм ще се обновява в група от интервали 3 сек, 3 сек, 3 сек, 1 сек. - нещо което лесно може да се забележи. Ако уеб сървърът се спре и се изчака малко, във вътрешния фрейм ще се види съобщение за грешка. Ако пак се стартира уеб сървърът след максимум до 10 сек., вътрешният фрейм ще се обнови и всичко ще продължи да работи нормално. Така чрез симулиране на проблем с мрежовата връзка, се демонстрира автоматичното продължаване на работа на разпределеното уеб приложение. Ако е необходимо вътрешният фрейм да не се вижда, може да се добави атрибут style="display:none" към тага iframe, т.е.:

```
<iframe src="frame.html" id="ifr" style="display:none">
```

Приложение 2. Пример за използване на стратегия "Дълго запитване"

В следващия пример е показано как се реализира техниката "дълго запитване" - "long polling". След получаване на заявката за вътрешния фрейм, уеб сървърът отлага, задържа връщането на отговор, докато не настъпи някакво събитие - в случая изтичане на интервал от време 7 сек. След зареждане на вътрешния фрейм, след 1 сек. се отправя нова заявка. Таймерът в родителския фрейм осигурява презареждане на вътрешния фрейм през 60 сек., в случай на мрежови проблеми.

Файл time_long_polling.js:

```
var http = require('http');
http.createServer(function (req, res) {
	res.writeHead(200, {'Content-Type': 'text/html'});
	if(req.url == '/' || req.url == '/index.html') {
		res.end('<html> \
			<head><title>Server Clock</title></head> \
			<body> \
				<div id="time">Clock</div> \
				<iframe src="frame.html" id="ifr"> \
				</iframe> \
				<script> \
					setInterval(function(){ document.getElementById("ifr").src
= document.getElementById("ifr").src;}, 60000); \
				</script> \
			</body> \
		</html> \
		');
		console.log('200: ' + req.url);
	} else if(req.url == '/frame.html') {
		setTimeout(function() {
```

```
                d = new Date();
                res.end('<html> \
                        <head><meta http-equiv="refresh" content="1" /></head> \
                        <body> \
                                <script> \
                                        parent.document.getElementById("time").innerHTML
= "' + d.toString() + '"; \
                                </script> \
                        </body> \
                </html> \
                ');
                console.log('200: ' + req.url);
        }, 7000);
    } else {
        res.end('<H1>ERROR 404</H1>');
        console.log('err404: ' + req.url);
    }
}).listen(1234, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1234/');
```

Приложение 3. Пример за използване на стратегия "Дълго запитване" от тип "push"

В примера по-долу е демонстрирана техниката "избутване" - "push". След получаване на заявката за вътрешния фрейм, уеб сървърът през 5 сек. започва да изпраща комбинация от отварящ таг, данни и затварящ таг (в случая тага script и програмен код). Браузърът, преди да е получил целия отговор (цялата уеб страница), започва да интерпретира таговете (да изпълнява програмния код). Това се повтаря 5 пъти, след което се задейства таймерът в родителския фрейм, който осигурява презареждане на вътрешния фрейм през 60 сек. Когато периодично се изпращат данните на порции, в началото допълнително се добавят безсмислени данни - пълнеж (в случая низ от интервали), така че общия обем на порцията данни да е по-голяма от 1 KB. В противен случай някои браузъри няма веднага да изпълнят изпратения им програмен код, а ще изчакат с интерпретацията докато не се получат поне 1 KB от уеб страницата или нейния край.

Файл time_push.js:

```
var http = require('http');
var s = " ";
for(i=0; i<10; i++) s += s;
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    if(req.url == '/' || req.url == '/index.html') {
        res.end('<html> \
            <head><title>Server Clock</title></head> \
            <body> \
                <div id="time">Clock</div> \
                <iframe src="frame.html" id="ifr"> \
                </iframe> \
                <script> \
```

```
                                setInterval(function(){ document.getElementById("ifr").src
= document.getElementById("ifr").src;}, 60000); \
                                </script> \
                        </body> \
                </html> \
                ');
                console.log('200: ' + req.url);
        } else if(req.url == '/frame.html') {
                res.write('<html><head></head><body>' + s); //данни >1KB
                var br = 0;
                var t = setInterval(function() {
                        d = new Date();
                        res.write('.<script> \
                                        parent.document.getElementById("time").innerHTML
= "' + d.toString() + '"; \
                                </script>\n \
                        ');
                        console.log(br);
                        br++;
                        if(br>5) {
                                clearInterval(t);
                                res.end('</body></html>');
                                console.log('200: ' + req.url);
                        }
                }, 5000);
        } else {
                res.end('<H1>ERROR 404</H1>');
                console.log('err404: ' + req.url);}
}).listen(1234, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1234/');
```

Приложение 4. Пример за използване на стратегия "AJAX"

В примера по-долу е показано как се работи с AJAX. В уеб страницата има бутон, при натискането на който асинхронно се изпраща заявка, без да се презарежда страницата. При получаване на отговор, той се поставя между HTML тагове и се прибавя към съдържанието на страницата. За разлика от фреймовете, където уеб сървъра трябва да изпрати HTML код, при AJAX се изпращат само данни и клиентската част на приложението поема изпълнението на процесите по оформяне и представяне по подходящ начин. С цел краткост на примера сме махнали голяма част от HTML таговете.

Файл ajax.js:

```
var http = require('http');
http.createServer(function (req, res) {
	res.writeHead(200, {'Content-Type': 'text/html'});
	if(req.url == '/' || req.url == '/index.html') {
		res.end(' \
			<div id="msg"> AJAX </div> \
			<script> \
				function sendRequest() { \
					xhr = new XMLHttpRequest(); \
					xhr.onreadystatechange = function() { \
						if(xhr.readyState==4 && xhr.status==200) \
						  document.getElementById("msg").innerHTML += 
"<p>" + xhr.responseText + "</p>"; \
					}; \
					xhr.open("GET", "ajax", true); \
					xhr.send(); \
				} \
			</script> \
			<input type="button" value="PING" onclick="sendRequest();" /> \
```

```
            ');
            console.log('200: ' + req.url);
        } else if(req.url == '/ajax') {
            d = new Date();
            res.end('PONG ' + d.toString());
            console.log('200: ' + req.url);
        } else {
            res.end('<H1>ERROR 404</H1>');
            console.log('err404: ' + req.url);
        }
}).listen(1234, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1234/');
```

Приложение 5. Пример за използване на стратегия "WebSocket"

Процедурите от страна на клиента за работа с обекта WebSocket, са следните:

```
//създаване на нов обект с URL, използващ протокол ws или wss (Secure WebSocket):
ws = new WebSocket("ws://example.com/test");
//функция, извиквана при осъществяване на мрежовата връзка:
ws.onopen = function() { alert("OPEN..."); };
//функция, извиквана при прекъсване на мрежовата връзка:
ws.onclose = function() { alert("CLOSE..."); };
//функция, която асинхронно се извиква при получаване на данни от сървъра - те се намират в полето data на параметъра:
ws.onmessage = function(e) { alert(e.data); };
//асинхронно изпращане на данни към сървъра:
ws.send("Test");
```

Работата от страна на сървъра е по-сложна и изисква повече познания за стандартите на WebSocket. Затова при програмиране на Node.js с използване на WebSocket и за клиентската, и за сървърната част се препоръчва използването на програмната библиотека socket.io.

Инсталиране на socket.io (http://socket.io/):

```
>npm install socket.io
```

Създаването на примерно приложение за онлайн текстова комуникация (текстов чат) по-долу е реализирано на няколко отделни етапа.

Файл chat.js:

```
var fs = require('fs');
var txt = fs.readFileSync('chat.html');

var server = require('http').createServer(function (req, res) {
```

```
res.writeHead(200); //оставяме типа на данните да е по подразбиране
res.end(txt);
});
server.listen(1234);

var io = require('socket.io')(server);
/*
всички HTTP заявки ще преминават първо през обекта io и после евентуално ще се предават на обекта server. Например HTTP заявките за URL /socket.io/socket.io.js ще се обслужват само от обекта io, а всички останали HTTP заявки - от обекта server.
*/
```

Файл chat.html:

```
<!doctype html>
<html>
	<head>
		<title>chat</title>
	</head>

	<body>
		<form action="">
			<input id="msg" autocomplete="off" />
			<button>Send</button>
		</form>

		<div id="txt"></div>

		<script src="/socket.io/socket.io.js"></script>
		<script>
			var socket = io();
```

```
        </script>
    </body>
</html>
```

Това представлява основната част, т.нар. "скелет" на уеб приложението. Следва тестване:

http://127.0.0.1:1234/socket.io/socket.io.js

Трябва да се зарежда код на JavaScript.

http://127.0.0.1:1234/ или http://127.0.0.1:1234/test

Трябва да се зарежда уеб страница с форма.

Сървърният модул socket.io, инсталиран при Node.js, включва в себе си клиентската библиотека socket.io.js. Като алтернатива може да се укаже клиентската библиотека на socket.io да се зарежда от мрежата за доставка на съдържание (Content Delivery Network, CDN) на socket.io:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.4/socket.io.js">
</script>
```

В този случай трябва изрично да се укаже и версията на библиотеката[1].

Като следващ етап от разработката се добавя програмен код за генериране на събития, които да водят до първоначално приветстване на потребителя.

В chat.js се добавя:

```
var io = require('socket.io')(server);
/* ... */
io.on('connection', function (client) {
    remote = client.request.connection.remoteAddress + ":" +
client.request.connection.remotePort;
    local = client.handshake.address.address + ":" + client.handshake.address.port;
```

[1] Информация за последните версии на клиентската библиотека се публикуват на (Socket.io, 2017).

```
    console.log("New connection from " + remote + " to " + local);

    client.emit('messages', { msg: 'System: Hello ' + remote + '!'});

});
```

В chat.html се добавя:

```
<script>
    var server = io();
    server.on('messages', function (data) {
        document.getElementById('txt').innerHTML += '<p>' + data.msg + '</p>';
    });
</script>
```

След заявка от браузъра и зареждане на уеб страницата, при браузъра се създава обекта io, който осъществява мрежова връзка със същия уеб сървър (същия хост и номер на порт), от който е заредена уеб страницата.

При сървъра се прихваща събитието за осъществена връзка, на конзолата се извеждат данни за клиента и се изпраща съобщение към клиента.

При клиента изпратеното съобщение генерира събитие, което се прихваща и в div елемента с идентификатор txt се прибавя текст - поздравление.

С метода .emit() се изпраща съобщение само до клиента, а с метода .broadcast.emit() се изпраща съобщение до всички клиенти, свързали се със сървъра и така на практика се реализира "избутване" - "push" на данни от сървъра към клиентите (виж фигурата).

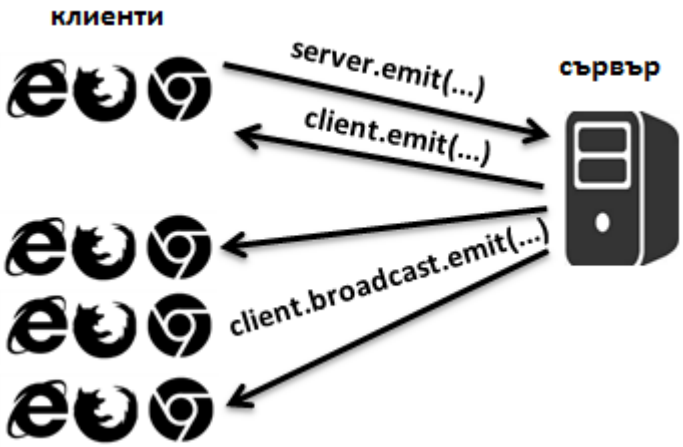


Схема на комуникацията между уеб клиенти и уеб сървъра при WebSocket.

Крайният вариант на чат-приложението без старите коментари е следния:

Файл chat.js

```
var fs = require('fs');
var txt = fs.readFileSync('chat.html');
var br = 0; //брояч за текущия брой на посетителите

var server = require('http').createServer(function (req, res) {
res.writeHead(200);
res.end(txt);
});
server.listen(1234);

var io = require('socket.io')(server);
io.on('connection', function (client) {
        br++;

        remote = client.request.connection.remoteAddress + ":" +
client.request.connection.remotePort;
        local = client.handshake.address.address + ":" + client.handshake.address.port;
```

```
        console.log("New connection from " + remote + " to " + local + " total:" + br);

        client.emit('messages', 'System: Hello ' + remote + '! Total: ' + br);
        client.broadcast.emit('messages', 'System: ' + remote + ' enter the chat. Total: ' +
br);

        client.on('messages', function(data) {
                console.log(remote + ':' + data);
                client.emit('messages', '<b>' + remote + ': ' + data + '</b>');
                client.broadcast.emit('messages', remote + ': ' + data);
        });

        client.on('disconnect', function(){
                br--;
                console.log(remote + " to " + local + " (" + br + ") disconnected");
                client.broadcast.emit('messages', 'System: ' + remote + ' leave the chat.
Total: ' + br);
        });
});
```

Файл chat.html:

```
<!doctype html>
<html>
        <head>
                <title>chat</title>
        </head>

        <body>
                <form action="" onsubmit="send(); return false;">
                        <input id="msg" autocomplete="off" />
```

```
            <button>Send</button>
        </form>

        <div id="txt"></div>

        <script src="/socket.io/socket.io.js"></script>
        <script>
            var server = io();
            server.on('messages', function (data) {
                document.getElementById('txt').innerHTML += '<p>' + data +
'</p>';
            });

            function send() {
                server.emit('messages',
document.getElementById('msg').value);
                document.getElementById('msg').value = '';
            }
        </script>
    </body>
</html>
```

По-долу е даден пример за приложение, позволяващо друг вид съвместна работа в реално време - едновременно рисуване, при което са използвани принципите, разгледани по-горе.

Файл draw.html:

```
<!doctype html>
<html>
<head>
    <title>Draw</title>
```

```
</head>

<body>
	<button type="button" onclick="color = 'red';">Red</button>
	<button type="button" onclick="color = 'green';">Green</button>
	<button type="button" onclick="color = 'blue';">Blue</button>
	<input type="color" onchange="color = this.value;" >
	0<input type="range" min="1" max="30" value="5" onchange="lw =
this.value;">30
	<canvas id="myCanvas" width="800" height="600" style="border: 1px solid
#000"></canvas>
<script>
		var x, y, isMouseDown = false, color = 'black', lw = 5;
		var canvas = document.getElementById('myCanvas');
		var context = canvas.getContext('2d');
		canvas.addEventListener('mousedown', function(evt) {
			isMouseDown = true;
		}, false);
		canvas.addEventListener('mouseup', function(evt) {
			isMouseDown = false;
		}, false);
		canvas.addEventListener('mousemove', function(evt) {
			x_old = x;
			y_old = y;
			x = evt.clientX - canvas.getBoundingClientRect().left;
			y = evt.clientY - canvas.getBoundingClientRect().top;
			if(isMouseDown) {
				server.emit('draw', [x_old, y_old, x, y, color, lw]);
			}
}, false);
```

```
</script>
	<script src="/socket.io/socket.io.js"></script>
	<script>
		var server = io();
		server.on('draw', function (data) {
			context.beginPath();
			context.moveTo(data[0], data[1]);
			context.lineTo(data[2], data[3]);
			context.strokeStyle = data[4];
			context.lineWidth = data[5];
			context.stroke();
			context.closePath();
		});
	</script>
</body>
</html>
```

Файл draw.js

```
var fs = require('fs');
var txt = fs.readFileSync('draw.html');
var br = 0;

var server = require('http').createServer(function (req, res) {
res.writeHead(200);
res.end(txt);
});
server.listen(1234);

var io = require('socket.io')(server);
io.on('connection', function (client) {
	br++;
```

```
        console.log("Total connections: " + br);

        client.on('draw', function(data) {
                client.emit('draw', data);
                client.broadcast.emit('draw', data);
        });

        client.on('disconnect', function(){
                br--;
                console.log("Total connections: " + br);
        });
});
```

Приложение 6. Пример за използване на стратегия "SSE"

В примера по-долу е демонстрирана работа с SSE. С цел ясно да се разграничат клиентската от сървърната част, са създадени два файла - уеб страница и уеб сървърно приложение. Показана е възможността за абониране към поток от събития.

Файл sse.html:

```
<input type="button" value="START" onclick="start();" />
<input type="button" value="STOP" onclick="stop();" />
<pre id="msg"></pre>
<script>
	var sse;
	var msg = document.getElementById("msg");
	function start() {
		if (!window.EventSource) {
			alert("No SSE");
			return;
		}
		sse = new EventSource("stream");
		msg.innerHTML = "\nSTART" + msg.innerHTML;
		sse.addEventListener(
			"event1",
			function(event) {
				msg.innerHTML = "\nevent1: " + event.data + msg.innerHTML;
			},
			false
		);
		sse.addEventListener(
			"message",
			function(event) {
				msg.innerHTML = "\ndata: " + event.data + msg.innerHTML;
```

```
                },
                false
            );
            sse.addEventListener(
                "error",
                function(event) {
                    if (event.target.readyState === EventSource.CLOSED) {
                        sse.close();
                        msg.innerHTML = "\nConnection closed!" +
msg.innerHTML;
                    } else if (event.target.readyState ===
EventSource.CONNECTING) {
                        msg.innerHTML = "\nConnection closed. Trying to
reconnect...\n" + msg.innerHTML;
                    } else {
                        msg.innerHTML = "\nConnection closed. Unknown error."
+ msg.innerHTML;
                    }
                },
                false
            );
        }

        function stop() {
            sse.close();
            msg.innerHTML = "\nSTOP" + msg.innerHTML;
        }
</script>
```

Файл sse.js:

```
var fs = require('fs');
var http = require('http');
http.createServer(function (req, res) {
	var t1, t2;
	if(req.url == '/' || req.url == '/index.html') {
		res.writeHead(200, {'Content-Type': 'text/html'});
		res.end(fs.readFileSync('sse.html').toString());
		console.log('200: ' + req.url);
	} else if(req.url == '/stream') {
		res.writeHead(200, {
			"Content-Type":"text/event-stream",
			"Cache-Control":"no-cache",
			"Connection":"keep-alive"
		});
		res.write("retry: 10000\n");

		t1 = setInterval(function() {
			res.write("data: " + (new Date()) + "\n\n");
		}, 1000);

		t2 = setInterval(function() {
			res.write("event: event1\n");
		}, 5000);

		req.connection.addListener("close", function () {
			clearInterval(t1);
			clearInterval(t2);
		}, false);
		console.log('SSE: ' + req.url);
	} else {
```

```
            res.end('<H1>ERROR 404</H1>');
            console.log('err404: ' + req.url);
        }
}).listen(1234, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1234/');
```

Приложение 7. Синхронни входно-изходни операции с двоични файлове в Node.js

Следната програма работи с двоичен файл, като записва 6 байта в него, след това прочита 5 байта и ги извежда на стандартния изход.

Файл sinc_binary.js:

```
var fs = require('fs'); // включване на библиотеката fs за работа с файлове

//Буфер за запис. Данните в масива са байтове.
wbuf = new Buffer([72, 101, 108, 108, 111, 33]);
try {
	fd = fs.openSync("data.bin", "w");
	fs.writeSync(fd, wbuf, 0, wbuf.length, 0);
	fs.closeSync(fd);
} catch (e) {
	console.log(e);
}

//Буфер за четене. Размерът му е фиксиран и не може да се променя.
rbuf = new Buffer(10);
try {
	fd = fs.openSync("data.bin", "r");
	fs.readSync(fd, rbuf, 0, 5, 0);
	fs.closeSync(fd);
} catch (e) {
	console.log(e);
}

console.log(rbuf);
```

След стартиране резултатът е следният:

```
>node sinc_binary.js
```

<Buffer 48 65 6c 6c 6f 00 00 00 00 00>

Класът Buffer е глобален клас[1] в Node.js и е подобен на масив от байтове. Не могат да се използват низове (обекти String), тъй като те използват формат UTF-8 и UTF-16, а те не са подходящи за работа с двоични данни. Флаговете при отваряне на файла са подобни на тези в езика C (r, w, a, r+, w+, a+) с някои нови моменти. Смисълът на аргументите на използваните функции за запис и четене са следните:

fs.writeSync(ФАЙЛОВ_УКАЗАТЕЛ, БУФЕР, ОТМЕСТВАНЕ_В_БУФЕРА, ДЪЛЖИНА, ПОЗИЦИЯ_ВЪВ_ФАЙЛА)

fs.readSync(ФАЙЛОВ_УКАЗАТЕЛ, БУФЕР, ОТМЕСТВАНЕ_В_БУФЕРА, ДЪЛЖИНА, ПОЗИЦИЯ_ВЪВ_ФАЙЛА)

, където:

ФАЙЛОВ_УКАЗАТЕЛ - получава се след отваряне на файла;

БУФЕР - масив от байтове;

ОТМЕСТВАНЕ_В_БУФЕРА - от коя позиция от масива да се прехвърлят данни;

ДЪЛЖИНА - колко байта да бъдат записани/прочетени;

ПОЗИЦИЯ_ВЪВ_ФАЙЛА - от коя позиция да започне операцията запис/четене.

Както се вижда, позиционирането в двоични файлове за операции от вида произволен достъп не се извършва с отделни функции или методи, като seek() или .fseek(), а със задаване на аргумент на методите fs.read() и fs.write(). Ако за аргумент се зададе null, операцията се извършва където е била предишната позиция.

За връщането на текущата позиция във файла, подобно на функции като tell() и ftell() в други езици, липсват функции и се налага програмистът сам да организира следенето на позицията.

[1] В документацията на Node.js се използва термина вградени класове, докато в документацията на JavaScript - вградени обекти, но смисъла е един и същ.

Приложение 8. Асинхронни входно-изходни операции с двоични файлове в Node.js

В следващия пример програмата от предходното приложение е пренаписана чрез използване на събития и изпълнението й е асинхронно.

Файл asinc_binary.js:

```
var fs = require('fs');

wbuf = new Buffer([72, 101, 108, 108, 111, 33]);
fs.open("data.bin", "w", function(err, fd) {
	if (err) throw err;
	fs.write(fd, wbuf, 0, wbuf.length, 0, function(err, written, wbuf) {
		if (err) throw err;
		fs.close(fd);
//… операции по четене от файла.
	});
});

//По-долния код трябва да се премести на мястото на горния коментар.
rbuf = new Buffer(10);
fs.open("data.bin", "r", function(err, fd) {
	if (err) throw err;
	fs.read(fd, rbuf, 0, 5, 0, function(err, bytesRead, rbuf) {
		if (err) throw err;
		fs.close(fd);
		console.log(rbuf);
	});
});
console.log("END");
```

Стартиране на програмата:

```
>node asinc_binary.js
```

```
END
<Buffer 48 65 6c 6c 6f 00 00 00 00 00>
```

За работа с двоични данни в Node.js има създадени множество библиотеки, улесняващи значително работата. Такива например са dissolve, binary-parser, binary-reader, bitparser и др. Как се извършва тяхното инсталиране е дадено малко по-нататък в главата.

Приложение 9. Обработка на данни в текстови файлове

Файл data.txt, в който се съхраняват идентификационен номер на туиит (tweet_id), харесвания (like_count) и брой ретуитове (retweet_count), като данните са във формат CSV:

```
759043035355312128,4567,297
10765432100123456789,1234,96
90071992547409921,987,55
```

Програма stat.js, която чете данните от файла и в нов файл "stat.txt" записва съотношението между брой харесвания и брой ретуитове с точност до втория знак:

```
var fs = require('fs');
var array = fs.readFileSync('data.txt').toString().split("\n");
var s = "";
for(i in array) {
	row = array[i].split(",");
	avg = (100*row[2]/row[1]).toFixed(2);
	s += row[0] + "," + avg + "\r\n";
}
fs.writeFileSync("avg.txt", s);
```

Съдържание на новосъздадения по програмен път текстов файл stat.txt:

```
759043035355312128,15.38
10765432100123456789,12.85
90071992547409921,17.95
```

Реализацията на програмата по-горе е с използване на синхронни функции.

Програма txtfile2array.js за синхронно и асинхронно прочитане съдържанието на текстовия файл data.txt и разполагане на редовете като елементи от масив:

```
var fs = require('fs');
```

```
//Синхронно
var buffer;
try {
	buffer = fs.readFileSync('data.txt');
} catch (e) {
	throw e;
}
var string = buffer.toString();
var array = string.split("\n");
//по-горното може да бъде написано и на един ред:
//var array = fs.readFileSync('data.txt').toString().split("\n");
for(i in array) {
	console.log(i + "\t" + array[i]);
}

//Асинхронно
fs.readFile('data.txt', function(err, data) {
	if(err) throw err;
	var array = data.toString().split("\n");
	for(i in array) {
		console.log(i + "\t" + array[i]);
	}
});
```

Методите .readFileSync() / .readFile() и .writeFileSync() / .writeFile() последователно отварят файла, извършват операциите четене или запис и затварят файла, т.е. при работа с тях не се отварят и затварят файлове като отделна операция. Ако четенето или записа трябва да се извършва на части може да се използват методите .createReadStream() / .createWriteStream().

При работа с текстови файлове като с двоични файлове трябва да се има предвид, че различните операционни системи използват различни символи за означаване на нов ред (line break, end of line), който при текстовите файлове има смисъла на "край на записа": в UNIX/GNU Linux и Mac OS X символът за нов ред е "\n" и е с ASCII код 10 (0x0A, 012, Ctrl-j, New line); в Windows се използва двойката символи "\r\n".

Приложение 10. Работа с програмна библиотека за работа с релационни бази от данни SQLite

В примера, който следва, се отваря за използване база от данни с име "test". Ако базата от данни не съществува, се създава нова. Базата от данни всъщност е един файл със същото име и затова то трябва да е съобразено с правилата за задаване на имена на файлове на операционната система, с която работим. В този файл се разполагат всички таблици и индексни файлове. Ако вместо име на базата от данни се зададе ":memory:" ще се създаде анонимна база от данни в паметта, която ще се изтрие след като се приключи работа с нея. Повече информация за имената на функциите и синтаксиса на SQL заявките може да се получи от документацията на модула (sqlite3 API, 2018) и от документацията на SQLite (Sqlite.org, 2018).

Файл sqlite3.js:

```
var sqlite3 = require('sqlite3');
var db = new sqlite3.Database('test');
db.serialize();
db.run("CREATE TABLE IF NOT EXISTS tweets(id INTEGER, msg TEXT)");

var q = db.prepare("INSERT INTO tweets VALUES (?, ?)");
for(var i=0; i<5; i++) q.run(i, "msg " + Math.random());
q.finalize();

db.parallelize();
db.each('SELECT * FROM tweets WHERE id<>random()', function(err, row) {
    if(row) console.log(row.id+"\t"+row.msg);
});
db.each('SELECT 1', function(err, row) {
    if(row) console.log(row);
});
```

```
db.close();
```

Резултат:

```
>node sqlite3.js
{ '1': 1 }
0       msg 0.057642508065328
1       msg 0.6009631438646466
2       msg 0.4358459026552737
3       msg 0.9632717058993876
4       msg 0.8242760335560888
```

Съществуват два начина за изпълнение на SQL заявките: последователно и успоредно във времето. По подразбиране всички заявки се изпълняват успоредно, т.е. заявката се предава за изпълнение и изпълнението на програмата продължава. При този случай, в зависимост от продължителността на изпълнение на заявките, е възможно заявки, които се срещат по-назад в програмния код да приключат изпълнението си преди тези, които се срещат по-напред.

При последователното изпълнение в даден момент от време се изпълнява само една заявка и чак когато изпълнението й приключи се преминава към изпълнението на следващата.

С функциите .serialize() и .parallelize() се превключва между последователно и успоредно изпълнение. В началото се иска заявките да се изпълняват в реда, в който са посочени - създаване на таблица, вмъкване на редове. В края има две заявки SELECT, които се изпълняват паралелно във времето и както се вижда от резултата, втората е приключила изпълнението си преди първата SQL заявка, която е по-сложна и изпълнението й отнема повече време[1].

[1] В конкретния пример коя от двете заявки ще приключи първа има до голяма степен вероятностен характер, тъй като обема на данните е много малък и времето за парсване на SQL заявката (която е символен низ) е съпоставимо с времето за изпълнение. При по-голяма

По-голяма и сложна таблица с първичен ключ, непразни и подразбиращи се стойности може да се създаде примерно така:

```
db.run("CREATE TABLE IF NOT EXISTS users (          \
        user_id INTEGER PRIMARY KEY,                \
        name varchar(100) NOT NULL DEFAULT '',      \
        comment TEXT                                \
)");
```

Поставянето на обратно наклонена черта в програмен код на JavaScript се налага, тъй като с нея се указва, че символният низ продължава на следващия ред.

сложност на първата заявка и по-голям обем данни, вероятността втората заявка да приключи първа нараства значително.

Приложение 11. Туит в JSON формат, използван за тестовете за скорост на преобразуване на низ в обект.


```
 { "created_at": "Tue Jan 01 22:44:03 +0000 2019",
   id: 1080233215615557600,
   id_str: '1080233215615557632',
   text:
    'Gas prices are low and expected to go down this year. This would be good!',
   truncated: false,
   entities: { hashtags: [], symbols: [], user_mentions: [], urls: [] },
   source:
    '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for
iPhone</a>',
   in_reply_to_status_id: null,
   in_reply_to_status_id_str: null,
   in_reply_to_user_id: null,
   in_reply_to_user_id_str: null,
   in_reply_to_screen_name: null,
   user:
    { id: 25073877,
      id_str: '25073877',
      name: 'Donald J. Trump',
      screen_name: 'realDonaldTrump',
      location: 'Washington, DC',
      description: '45th President of the United States of America????',
      url: 'https://t.co/OMxB0x7xC5',
      entities: [Object],
      protected: false,
      followers_count: 56800570,
      friends_count: 45,
      listed_count: 98291,
```

created_at: 'Wed Mar 18 13:46:38 +0000 2009',
favourites_count: 7,
utc_offset: null,
time_zone: null,
geo_enabled: true,
verified: true,
statuses_count: 40116,
lang: 'en',
contributors_enabled: false,
is_translator: false,
is_translation_enabled: true,
profile_background_color: '6D5C18',
profile_background_image_url:
'http://abs.twimg.com/images/themes/theme1/bg.png',
profile_background_image_url_https:
'https://abs.twimg.com/images/themes/theme1/bg.png',
profile_background_tile: true,
profile_image_url:

'http://pbs.twimg.com/profile_images/874276197357596672/kUuht00m_normal.jpg',
profile_image_url_https:

'https://pbs.twimg.com/profile_images/874276197357596672/kUuht00m_normal.jpg',
profile_banner_url:
'https://pbs.twimg.com/profile_banners/25073877/1543104015',
profile_link_color: '1B95E0',
profile_sidebar_border_color: 'BDDCAD',
profile_sidebar_fill_color: 'C5CEC0',
profile_text_color: '333333',
profile_use_background_image: true,

has_extended_profile: false,
default_profile: false,
default_profile_image: false,
following: false,
follow_request_sent: false,
notifications: false,
translator_type: 'regular' },
geo: null,
coordinates: null,
place: null,
contributors: null,
is_quote_status: false,
retweet_count: 20235,
favorite_count: 124248,
favorited: false,
retweeted: false,
lang: 'en' }

Приложение 12. Програма, използвана при тестовете за производителност при обработка на данни във формат JSON при използване на различни програмни средства.

```
<!DOCTYPE html>
<html><head></head><body><pre>
<script>
const REP = 10000;
json_str = '{"created_at":"Tue Jan 01 22:44:03 +0000
2019","id":1080233215615557600,"id_str":"1080233215615557632","text":"Gas prices
are low and expected to go down this year. This would be
good!","truncated":false,"entities":{"hashtags":[],"symbols":[],"user_mentions":[],"urls":[]
},"source":"Twitter for
iPhone","in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_
id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":2507
3877,"id_str":"25073877","name":"Donald J.
Trump","screen_name":"realDonaldTrump","location":"Washington,
DC","description":"45th President of the United States of
America????","url":"https://t.co/OMxB0x7xC5","entities":[null],"protected":false,"followe
rs_count":56800570,"friends_count":45,"listed_count":98291,"created_at":"Wed Mar
18 13:46:38 +0000
2009","favourites_count":7,"utc_offset":null,"time_zone":null,"geo_enabled":true,"verifi
ed":true,"statuses_count":40116,"lang":"en","contributors_enabled":false,"is_translator
":false,"is_translation_enabled":true,"profile_background_color":"6D5C18","profile_bac
kground_image_url":"http://abs.twimg.com/images/themes/theme1/bg.png","profile_ba
ckground_image_url_https":"https://abs.twimg.com/images/themes/theme1/bg.png","p
rofile_background_tile":true,"profile_image_url":"http://pbs.twimg.com/profile_images/8
74276197357596672/kUuht00m_normal.jpg","profile_image_url_https":"https://pbs.twi
mg.com/profile_images/874276197357596672/kUuht00m_normal.jpg","profile_banner
_url":"https://pbs.twimg.com/profile_banners/25073877/1543104015","profile_link_colo
r":"1B95E0","profile_sidebar_border_color":"BDDCAD","profile_sidebar_fill_color":"C5
```

```
CEC0","profile_text_color":"333333","profile_use_background_image":true,"has_exten
ded_profile":false,"default_profile":false,"default_profile_image":false,"following":false,"
follow_request_sent":false,"notifications":false,"translator_type":"regular"},"geo":null,"c
oordinates":null,"place":null,"contributors":null,"is_quote_status":false,"retweet_count":
20235,"favorite_count":124248,"favorited":false,"retweeted":false,"lang":"en"}';
data = new Array();
i = REP;
while(i-->0) {
//      data.push(json_str.replace("2019", i));
        data.push(json_str);
}

bench(f1); bench(f2); bench(f3); document.writeln("\n");
bench(f1); bench(f2); bench(f3); document.writeln("\n");
bench(f1); bench(f2); bench(f3); document.writeln("\n");
bench(f1); bench(f2); bench(f3); document.writeln("\n");
bench(f1); bench(f2); bench(f3);

function f1(json_str) {
        return JSON.parse(json_str);
}

function f2(json_str) {
        return (new Function("return " + json_str))();
}

function f3(json_str) {
        return eval("(" + json_str + ")");
}
```

```
function bench(func) {
        start = new Date(); start = start.getTime();
        data.forEach(func);
        end = new Date(); end = end.getTime();
        document.writeln(Math.round(1000*REP/(end-start)));
}

</script>
</pre>
</body></html>
```

При варианта, предназначен за Node.js, промените са минимално необходимите и са следните: първо, премахнати са HTML таговете, като е оставен само програмния код между <script>...</script>. Второ, вместо document.writeln() е използвано console.log().

УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

ДЕКЛАРАЦИЯ

От доц. д-р Павел Стоянов Петров

(име, презиме и фамилия на кандидата за придобиване на ОНС „доктор“ или кандидата за придобиване на научната степен „доктор на науките“)

Декларирам, че:

1. Представеният дисертационен труд е подготвен и изпълнен самостоятелно от мен.
2. В дисертационния труд не са използвани пряко или косвено чужди текстове или, ако са използвани части от такива, те са позовани / цитирани и никоя част от дисертационния ми труд не е в нарушение на авторските права на институция или личност.
3. Никоя част на дисертационния ми труд не е бил представян в този вид в същата или в друга образователна и/или научна институция за присъждане на образователна или научна степен. При констатиране на несъответствие с декларираните от мен обстоятелства по точки 1, 2 и 3 от настоящата декларация, нося отговорност в съответствие със Закона и нормативните документи на УниБИТ.

Дата: 05.11.2021 г. **Подпис**.....