



Технически Университет София

ДИПЛОМНА РАБОТА

Уеб-базирано приложение за търсене на
зависимост на поведение на потребители
от лог файл

Разработил:

Васил Димитров Димитров
Факултетен номер: 501214037
Административна група: 51

Научен Ръководител:

Доц. д-р инж. Аделина Алексиева-
Петрова

Съдържание

1. Увод.....	4
2. Функционални изисквания	6
2.1. Функционалностите на потребител.....	6
2.2. Кой ще използва продукта.....	6
2.3. Какви данни се използват	7
2.4. Как се достъпват функционалностите от потребителя.....	7
2.5. Алгоритми за търсене на зависимост на поведение на потребители	7
2.5.1. Алгоритъм LCM	8
2.5.2. Алгоритъм RP-Growth.....	9
2.5.3. Алгоритъм VTSA	12
3. Проектиране на софтуерната система	13
3.1. Среда.....	13
3.2. Технически изисквания.....	13
3.2.1. Java	13
3.2.2. Gradle	14
3.2.3. MariaDB	14
3.2.4. Spring	14
3.2.5. Lombok	15
3.3. База данни	16
3.4. Проектиране на софтуерната архитектура.....	16
4. Реализация на системата	18
4.1. Реализация в сорс кода на приложението	20
4.2. Реализация на потребителския интерфейс	31
5. Ръководство за работа със системата	32
5.1. Качване на лог файл	32

5.2. Преглед на резултати	35
5.3. Конфигурация на алгоритмите за анализ	37
5.3.1. Често достъпвани събития	39
5.3.2. Редки събития	40
5.3.3. Седмичен времеви трафик	42
5.4. Достъп до кратко ръководство от сайта	44
6. Заключение	46
7. Литература	47
7.1. Използвани външни библиотеки:	47
7.2. Използвани онлайн ресурси:	47
8. Приложение	50

1. Увод

В днешно време почти всеки софтуер работещ с клиенти пази по някакъв начин техните дейности, било то чрез лог файл или чрез база данни, с цел извличане на полезна статистическа информация. В първите етапи на развитие на един такъв софтуер тези анализи могат да се извършват сравнително лесно от самите служители, например чрез използването на табличен софтуер като microsoft excel. Но с течение на времето и с нарастване на броя клиенти, тези съхранявани данни бързо нарастват до размери непосилни за обработка и анализ през обичайните и познати на всички приложения, тогава идва на помощ data mining.

Data Mining накратко представлява процес на откриване на смислени корелации, зависимости, повтарящи се образци, тенденции и аномалии в големи масиви от данни чрез използването на специализирани алгоритми.

Този процес се използва главно от компании със силен фокус върху оптимизиране и подобряване на техните услуги спрямо клиентските нужди и поведение. Използвайки data mining позволява не само анализ на минали действия, но и предвиждане на бъдещи такива.

За това е разработена системата DataMiner. Системата представлява уеб-базирано приложение за намиране зависимост на поведение на потребители от лог файл чрез реализирането на 3 различни алгоритми за извличане на данни.

Всеки един потребител с достъп до системата има възможност да качи лог файл във формат .csv за да се извършат съответните анализи. Резултатите се визуализират в стилистичен вид, но могат да бъдат запазени чрез експортиране в текстови файл, в който резултатите не са така красиви, но са по-подробни. Съществува и възможност за промяна на различните настройките за използвани алгоритми, както и опция за изключване на даден алгоритъм от крайния резултат с цел забързване на анализа.

Системата разполага с минималистичен и улеснен съвременен дизайн, който работи с мобилни устройства (responsive дизайн). Това дава мултифункционалност на системата, като възможност за работа през различни устройства.

2. Функционални изисквания

Функционалните изисквания се отнасят до това какво системата трябва да може да прави. При функционалните изисквания, трябва да се знае какво прави системата и какво прави човека. Взимането на решения относно това кои действия се изпълняват от човека и кои от компютъра (известно като процес на разпределяне на задачите) се отлага до процеса на анализиране на задачите.

Определянето на функционалните изисквания на дадена система се случва на ниво проектиране на самата система, за да се определи какви ще бъдат функционалностите, които системата ще поддържа и да се знае какво трябва да бъде разработено.

2.1. Функционалностите на потребител

- Качване на лог файл
- Свлячане на файл с анализи
- Редактиране настройките на алгоритмите

2.2. Кой ще използва продукта

Приложението е разработено за собственици на сайтове/софтуери с голям трафик, за които трябва да се извлече някакъв статистически анализ, било то с цел по-добро планиране в кои ресурси да се инвестира или за вземане на решение, кога системата може да бъде спирана за профилактика.

Също така приложението със своя отворен код може да помогне на студенти/разработчици, които са заинтересувани как да имплементират съответните алгоритми в Java код.

2.3. Какви данни се използват

За да се използва основната функционалност на системата, потребителят трябва да разполага с лог файл в правилен формат и да го качи в приложението. Пример за това как изглежда правилния формат на лог файла може да се види на „Помощ“ страницата на приложението.

Основни валидации:

- Валидация за празен лог файл
- Валидация за разширение на лог файл
- Валидация за правилен формат на лог файл
- Валидация за липсващ файл с анализи
- Валидация за празни данни и неправилни стойности при редакция на настройките за анализ

2.4. Как се достъпват функционалностите от потребителя

Всеки потребител има достъп до всички функционалности на приложението.

В бъдеща реализация ще се добави функционалност за регистриране, тогава алгоритмичните настройки ще са различни за всеки логнат потребител, а нерегистрираните потребители ще използват същите стандартни настройки без право за редакция.

2.5. Алгоритми за търсене на зависимост на поведение на потребители

Основната функционалност на приложението, извличането на зависимости между поведението на потребителите, се дължи на имплементацията на три алгоритъма в програмния код.

2.5.1. Алгоритъм LCM

Намирането на често достъпвани елементи е един от основните проблеми в областта на извличането на данни, тъй като могат да се приложат за извличане на правила за асоцииране, индуктивни бази данни и разширяване на заявки. За правилното функциониране при голям обем данни са необходими бързи реализации. Една такава реализация е алгоритъма LCM, който е победителят в конкурса FIMI (Frequent Itemset Mining Implementations) състоял се през 2004г. Предполага се, че той е един от най-бързите алгоритми за намиране на затворени чести множества от събития. За да се обясни смисъла на израза затворени чести множества, първо трябва да се изяснят няколко други:

- Подкрепа – това е броят на потребителски сесии достъпили даденото множество от събития. Ако имаме 10 потребителя и 4 от тях са достъпили даденото множество, то се води че е с подкрепа 0,4 или 40%.
- Чести множества – множества, които са били достъпени поне или повече от предварително определен минимален брой подкрепа.
- Затворени чести множества – това са чести множества, които не присъстват в друго надмножество от чести множества със същата подкрепа.

LCM използва префикс (наречен pps), запазващ разширение от едно затворено множество към друго. Разширението индуцира дърво за търсене от множеството често затворени набори от елементи, като по този начин можем напълно да изброяваме затворени набори от елементи без дублиране. Генерирането на този префикс не се нуждае от предварително получено затворено множество, следователно използването на памет от LCM не зависи от броя на затворени множества.

Времевата сложност на LCM се изразява в линейна функция спрямо броя на затворените множества от елементи.

Генерирането на ррс разширения не зависи от сложна структура от данни, каквато например е бинарното дърво. LCM е рализиран само с масиви, което го прави бърз и превъзхождащ спрямо други подобни алгоритми.

Недостатък на този алгоритъм е, че няма функция за намаляне размера на базата данни по време на изпълнение. Имплементацията му в приложението е реализирана с добавена такава функционалност, правейки LCM по-оптимизиран.

2.5.2. Алгоритъм RP-Growth

Повечето алгоритми за извличане на данни имат за цел намиране на чести корелации. Редките асоциации обаче в някои случаи са по-интересни от честите, тъй като редките елементи представляват неочаквани или непознати асоциации. Всички текущи алгоритми за добив на редки асоциации използват в основата си подхода на „Apriori”, който има изчислително скъпи стъпки за генериране и подрязване на кандидатите. Точно поради тази причина в RP-Growth се използва “RP-Tree” (Rare Patter Tree), представляващ метод за извличане на подмножество от редки правила за асоцииране с помощта на дървовидна структура и компонент за получаване на информация, който помага да се идентифицират по-интересните правила за асоцииране.

Техниките за намиране на асоциации се използват за извличане на полезна информация от бази данни. Наборът от асоциирани данни, който може да бъде извлечен от база данни, може да бъде разделен чрез праг на подкрепа на чести и редки. И честите и редките представят различна информация за базата данни, от която са извлечени, тъй като честите правила се фокусират върху модели, които се срещат често, докато редките правила се фокусират върху модели, които се срещат рядко. В много области събитията, които се случват често, могат да бъдат по-малко интересни от събитията, които се случват рядко, тъй като честите модели представляват изветни и очаквани резултати, докато редките модели могат да представляват неочаквани или неизвестни досега асоциации, което е полезно за експертите в съответните области. Например в областта на медицината очакваните, чести реакции на

лекарствата са по-малко интересни от изключителните, редки реакции, които могат да показват нежелани реакции или лекарствени взаимодействия.

Алгоритми като Apriori могат да бъдат използвани за намиране както на чести, така и на редки асоциации, но последните изискват минималният праг на подкрепа да бъде зададен на ниска стойност. Това обаче може да доведе до комбинаторни експлозии от набори от елементи, тъй като броят на моделите, които отговарят на минималната подкрепа нараства драстично при по-голям набор от данни. Като се имат предвид n артикула, броят на възможните набори от елементи е $2^n - 1$.

Има три възможни типа редки набори от елементи:

- първо, набори от елементи, които се състоят само от редки елементи;
- второ, набори от елементи, които се състоят от редки и чести елементи;
- трето, наборите от елементи, които се състоят само от чести елементи, които падат под минималния праг на подкрепа.

Ние наричаме набори от елементи от първия и втория тип като групи от редки елементи. Наборите от елементи на тези елементи обикновено са по-интересни от наборите от третия тип, които наричаме набори от елементи с не редки елементи. Това е така, защото често срещаните елементи се срещат често в базата данни и може да има много набори от нередки елементи, които не представляват интересна връзка между елементи, тъй като елементите са възникнали заедно случайно.

Примерни за твърдението, че наборите от редки асоциации могат да бъдат по-важни:

Да предположим, че база данни със симптоми на пациента съдържа редките набори от елементи „1: {повишена сърдечна честота, висока температура, синини по кожата, ниско кръвно налягане}” и „2: {мускулна болка, шум в ушите, кихане, киселини в стомаха“, където всички симптоми, различни от „ниско кръвно налягане“, са чести елементи. Набор от елементи

1 е набор от асоциации с редки елементи, а набор от елементи 2 е набор от не-редки елементи. Набор 1 съдържа подмножество от симптомите на сепсис, което ще създаде правило като „{повишен сърдечен ритъм, температура, синини по кожата } → ниско кръвно налягане“, което подчертава връзката между различните три предишни симптома с ниско кръвно налягане, което е симптом на тежък сепсис. За разлика от това, данните, генерирани от набор 2, като "{мускулна болка, шум в ушите, киселини в стомаха} → кихане" не дават никаква полезна информация, тъй като всички тези симптоми са често срещани поотделно и просто са се появили заедно случайно.

В приложението Data Miner е имплементиран алгоритъма, наречен RPGrowth, който открива набори от редки елементи, използвайки дървовидна структура. За разлика от предишни алгоритми с подход анализиране по нива, RPGrowth не е необходимо да генерира и тества всички правдоподобни комбинации от редки набори от елементи, което е по-ефективно. RPGrowth е адаптация на RP_Tree алгоритъма, който е първият рядък алгоритъм за добив на правила за асоцииране, който използва дървовидна структура.

2.5.3. Алгоритъм VTSA

За разлика от предходните алгоритми, VTSA не търси нито чести, нито редки елементи. Той извършва анализ на транзакциите в бази данни спрямо това, в кои дни и часови периоди от седмицата те са били изпълнени. Това го прави по-специализиран в отношение на формат на входни данни, което означава че има приложимост в много по-конкретни случаи, но това не го прави по-малко важен. От върнатия анализ на данните може да се разбере кога системата е най-много, и кога най-малко натоварена. Тази информация е полезна в редица случаи, особено в сферата на онлайн приложенията, където е добре периодично да се планира спиране на системата за поддръжка, което е добре да става в най-малко натоварените периоди, или по-добро разпределение на процесорни ресурси, в по-натоварените периоди.

При стартиране VTSA създава масив от обекти с размер $7 \times (24/X)$, където X е предварително зададения желан часови интервал от потребителя. Обектите пазят информация за това кой ден и кой часови интервал представляват, както и брой на срещнати транзакции които влизат в този диапазон. Следователно използването на памет от VTSA не зависи от броя на транзакции, а единствено от въведения часови интервал. В приложението е въведена долна граница от 4 за часовия интервал, за да се избегне драстичното нарастване на сложност и време за анализ свързано с намалянето на интервала.

3. Проектиране на софтуерната система

3.1. Среда

Като среда за разработка на системата ще се използва Eclipse IDE. Това е многоезична среда за разработване, която включва интегрирана среда за разработка (IDE) и плъгин система. Написана е главно на Java и може да бъде използвана за разработване на приложения на Java, а с помощта на различни плъгини- и на различни други езици. Поддържа и разработката на уеб сайтове чрез Java EE.

3.2. Технически изисквания

- Като език за разработката на системата, ще се използва Java 11.
- За изграждане на системата и използването на библиотеки, ще се използва Gradle.
- Базата данни ще бъде на MariaDB, като приложението ще използва Spring Data за връзка с базата данни.
- За интерфейса, ще се използва Thymeleaf, като репрезентация на подадените обекти от страна на сървъра.
- Шаблонът за проектиране, който ще се използва е MVC – Model View Controller.

3.2.1. Java

Java е програмен език и компютърна платформа, която е реализирана за пръв път през 1995г. от Sun Microsystems. Вече съществуват много приложения и уеб сайтове, които не биха могли да работят без Java. Java е бърз, сигурен и надежден начин за реализиране на идеи. Чрез използването на Java, могат да бъдат реализирани различни видове приложения и системи, като конзолни приложения, уеб сайтове, мобилни приложения, десктоп приложения и други. Виртуалната машина на Java позволява приложенията да бъдат изпълнявани на различни платформи без да е нужна промяна по системата.

Чрез използването на Java е възможно разработването и управлението на бизнес софтуер, включително мрежови и уеб услуги в голям мащаб, които са сигурни и надеждни. Java EE е разширеният вариант на Java SE.

Осигурява API за обектно-релационните планирания, многослойни архитектури и уеб услуги. В платформата се включват главно дизайнерски компоненти, работещи на сървър. Софтуерът за Java EE е написан основно на езика Java.

3.2.2. Gradle

Gradle е автоматичен build инструмент, който се използва главно върху Android и Java проекти. Той описва как дадения проект ще се build-ва и кои библиотеки участват в изграждането на проекта. Конфигурацията се пише чрез езика Groovy във специфичен файл с името build.gradle. Чрез Gradle могат да бъдат добавени различни плъгини, които да извършват различни действия при изпълнението на build процеса.

3.2.3. MariaDB

MariaDB е разклонена разработка от MySQL базата данни, като тя също е релационна и е с безплатен лиценз под GNU GPL. Разработена е от някои от оригиналните разработчици на MySQL, които са я разклонили след сливането ѝ с Oracle. MariaDB е напълно съвместима с MySQL и включва всички нейни спецификации и функции.

3.2.4. Spring

Spring Framework е framework с отворен код (open source) за Java платформата. Spring framework предоставя много функции, които улесняват разработването на Java-базирани enterprise приложения. Spring Framework включва различни модули, които предоставят широк набор от

функционалности. Spring е неразделна част от уеб разработките с Java. Списък от възможности за използване на модули от Spring е голям, като например Spring MVC, Spring Security, Spring Cloud и много други.

3.2.5. Lombok

Програмния език Java е известен със сравнително по-обемистия си код спрямо много други езици за програмиране. В следствие на това е разработена библиотека, която спестява писането на код за неща, които не е нужно да се изписват всеки път, а могат да се генерират. Тази библиотека се казва Lombok, тя премахва нуждата от писането на еднотипен генериран код и по този начин се вдига продуктивността и бързодействието на програмиста. Нещата, които премахва и облекчава всеки програмист на Java са getters/setters, конструктори. Това са неща, които са с висока повтораемост в писането на Java приложения. Чрез Lombok също така могат да бъдат генерирани автоматично hashCode и equals методи за даден обект, както и да се добави logger чрез проста анотация.

3.3. База данни

Схемата на базата данни е сравнително проста, състояща се само от една таблица `algo_settigns` за пазене на стандартните алгоритмични настройки, които потребителя може да променя. На фиг. 1 е показана структурата на единствената таблица с базата:

DATA_MINER: algo_settings

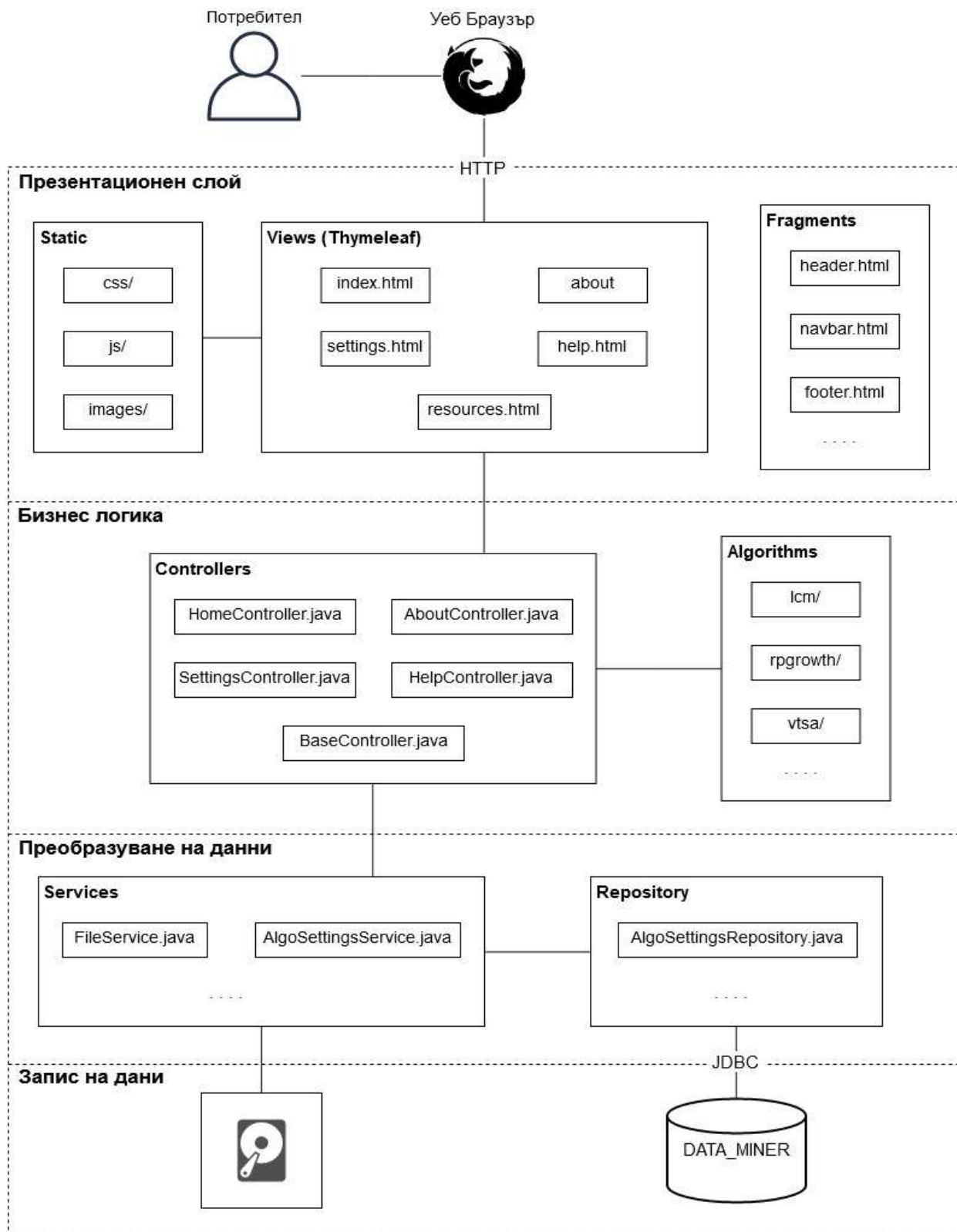
#	Name	Datatype
1	id	INT
2	lcm	TINYINT
3	lcm_min_sup	DOUBLE
4	rpg	TINYINT
5	rpg_min_rare_sup	DOUBLE
6	rpg_min_sup	DOUBLE
7	vtsa	TINYINT
8	vtsa_hour_interval	INT

В бъдеща реализация ще бъдат добавена таблица `user`, както и колона `user_id` в таблица `algo_setings`, които ще се използват за регистрация и пазене на персонални настройки за всеки потребител.

3.4. Проектиране на софтуерната архитектура

Всички използвани технологии и ресурси комуникират помежду си по възможно най-оптималния начин, като е гледано да се запази модулността на отделните компоненти, така че да могат да бъдат подменяни с минимални усилия и промени в кода.

Как различните компоненти комуникират помежду си:



4. Реализация на системата

Уеб приложението съдържа модерен и улеснен дизайн използващ комбинация от Bootstrap CSS и custom CSS. Дизайнът позволява работата със системата да бъде извършвана на други устройства, като смартфон и таблет, тъй като дизайна е responsive и при по-ниска резолюция на екрана, елементите се подреждат и оразмеряват спрямо новата резолюция по такъв начин, че да пасва на работата с новото устройство. Това е полезно в днешни дни, тъй като голяма част от трафика в интернет е от мобилни устройства.

Проекта е разработен на Java 11, като се използва Gradle за инструмент за изграждане. Всички нужни библиотеки се описват в build.gradle-a.

Проекта използва Spring boot версия 2.1.8.RELEASE за инициализиране на биновете нужни при Dependency Injection. Всичко е конфигурирано автоматично и така няма нужда да се използват xml конфигурационни файлове, правейки кода по-лесен за разбиране. Използва се component-scan, като по този начин може да се използва анотацията @Autowired за автоматично инжектиране на бина.

Чрез Thymeleaf са конфигурирани слоевете, които да бъдат преизползвани, като хедъра, навигационното меню и футера с информация за автора.

Използва се Slf4j за логване на грешки, като се комбинира с анотацията @Slf4j от библиотеката Lombok за да не се налага да се инициализира логър обект за всеки клас. Конфигурациите за ниво на логване, локация и име на лог файл са добавени в application.properties файла.

Проекта използва CrudRepository интерфейса на Spring Data за връзка с базата. Така всички CRUD операции могат да се използват, без да се пише никакъв допълнителен код.

DataSource конфигурацията към базата се намира в файла application.properties, тъй като Spring Boot идва с вграден Tomcat сървър, така че няма нужда от използването на JNDI.

Шаблонът по който е имплементирана базата в Java приложението, е Entity, Repository, Service. Entity представлява таблицата или модела, в Repository-то се пишат всички дефинирани заявки към базата, а Service извиква Repository методите, като самия Repository клас се инжектира чрез @Autowired анотацията. Ако приложението има нужда от информация от базата, в дадения контролер се инжектира съответния Service клас (@Autowired отново) и по този начин се прави заявка към базата.

Използваната база данни е MariaDB с версия 10.3.10 с енкодинг UTF8, което позволява съхранението на данни на кирилица, както и на специални символи.

Базата данни се стартира автоматично при включване на компютъра чрез създаденият windows service.

За сървър на приложението се използва вградения в Spring Boot Tomcat. Единствена допълнителна конфигурация която е добавена е за промяна на порта за достъп от 8080 на 8000. При стартиране на приложението сървърът стартира сам, а приложението автоматично се качва на Tomcat.

4.1. Реализация в сорс кода на приложението

Тъй като е използван Spring Boot в проекта, няма нужда от конфигурация на анотации и сканиращата част за компоненти, където се декларира пакети в приложението, които да сканира и търси за анотации от Spring. Единственото нужно за стартиране на Spring Boot приложението е `SpringBootApplication` класът, което прави кодът да изглежда сравнително по-малък и елегантен. На класът с `main` методът се слага `@SpringBootApplication` анотацията която включва автоматичните Spring Boot конфигурационни механизми, включва компонентния scanner на всички пакети които са на същото ниво като `Application` класът, като също така позволява и допълнителни конфигурации да се добавят от програмиста използвайки `@Configuration` анотацията. В случай, че имаме пакет с бийнове, който е на по-горно ниво от `Application` класът, може да се добави параметър (`scanBasePackages = "package"`) към `@SpringBootApplication`, което ще укаже на компонентния scanner да мине и през споменатия пакет.

Пример от гореспоменатия текст:

```
/**
 * Main class of DataMiner
 *
 * @author Vasil.Dimitrov^2
 */
@SpringBootApplication(scanBasePackages = {"package_1", "package_2"})
public class DataMinerApplication {

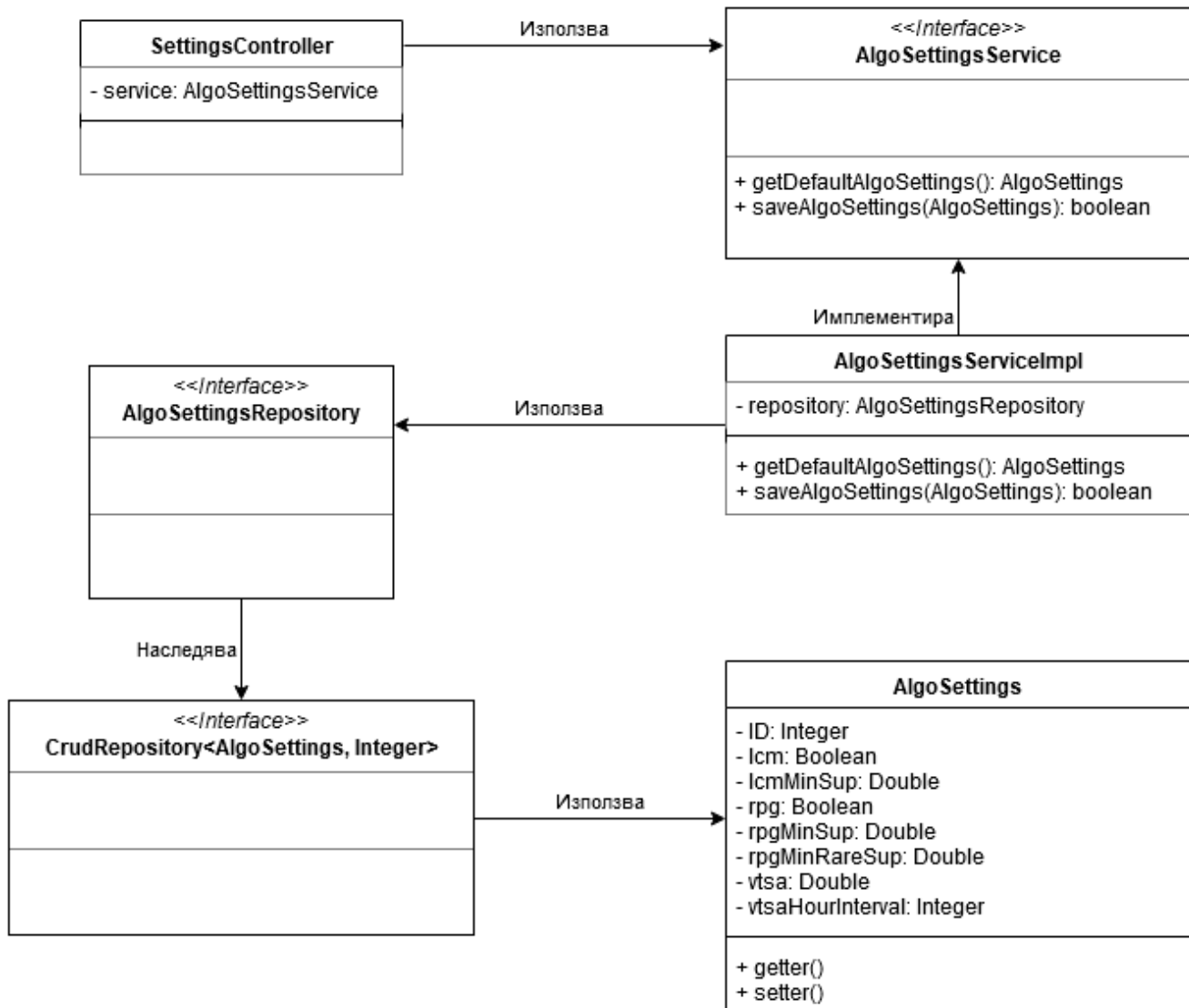
    public static void main(String[] args) {

        SpringApplication.run(DataMinerApplication.class, args);

    }

}
```

За заявки към базата данни се използва Entity, Repository и Service. На изображението по-долу е показана клас диаграма, която използва описания модел.



- Entity клас

Моделът представлява репрезентация на таблицата в java обект. Използват се няколко основни анотации от Javax Persistence и Lombok за дефинирането им.

Анотация Data е от библиотеката Lombok. Тя премахва нуждата от писането на getter-и, setter-и, празен конструктор, конструктор за всички задължителни полета, също така пренаписва методите toString, equals и hashCode.

Анотация @Entity дефинира, че обекта ще бъде репрезентация на таблица от базата данни. Като аргумент може да се подаде името на колоната, която ще бъде репрезентирана.

Анотация @Column служи за свързване на поле с колона от таблица в базата данни. Чрез подадени аргументи може да се дефинират името на колоната, типа на колоната, дали може да е празна и дали данните в нея трябва да са уникални спрямо таблицата. Не е задължителна.

Анотацията @Id служи за указване на primary key колона. Тя е задължителна, без нея entity-то не работи.

Анотацията GeneratedValue се използва за указване, че съответната колона ще се популира от автоматично генерирани стойности, които в повечето случаи са поредни числа започващи от 1. Чрез аргумент може да се посочи различен начин на генериране на тези стойности.

```
@Data
@Entity(name = "algo_settings")
public class AlgoSettings {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(updatable=false, nullable=false, unique=true)
    private Integer ID;

    @Column(nullable=false, columnDefinition="TINYINT(1)")
    private Boolean lcm;

    ....
}
```

- Repository клас

В Repository-то се дефинират всички заявки, като отделни методи, към дадената таблица. Наследяването на интерфейс CrudRepository предоставя всички CRUD операции към таблицата без да е нужно да ги пишем, като в същото време указва на Spring, че класът трябва да се инжектира като компонент в слоя за услугата, без да е нужно да се добавя анотация @Repository. Така ако не са нужни по-конкретни заявки кодът на приложението остава малък, тъй като repository класът остава празен.

```
public interface AlgoSettingsRepository extends  
CrudRepository<AlgoSettings, Integer> {  
  
}
```

- Service клас

Всеки клас услуга (Service) трябва да бъде има анотацията @Service от Spring Framework библиотеката. Тя се дефинира, за да може да се използва @Autowired чрез интерфейса на услугата и по този начин да се инжектира имплементация без да е нужна нова инициализация.

След това обикновено се добавя анотацията @Transactional отново на Spring Framework библиотеката, въпреки че не е задължителна. Чрез нея се указва, че класът ще дефинира транзакции към базата. Чрез аргументите на тази анотация се описва дали ще се извършва само четене или и други операции като добавяне, изтриване или промяна.

За инжектиране на repository към класът трябва да се използва анотацията @Autowired, без нея repository-то ще бъде NULL и всякакви негови извиквания биха довели до грешки.

```
@Service
@Transactional(readonly = false)
public class AlgoSettingsServiceImpl implements AlgoSettingsService {

    private final AlgoSettingsRepository repository;

    @Autowired
    public AlgoSettingsServiceImpl(AlgoSettingsRepository repository) {
        this.repository = repository;
    }

    @Override
    public AlgoSettings getDefaultAlgoSettings() {
        return this.repository.findById(Constant.DEFAULT_ALGO_SETTINGS_ID)
            .orElse(null);
    }

    . . . .
}
```


- Controller клас

Контролера е част от MVC шаблона, който Spring използва. Контролерът е отговорен за обработката на потребителските заявки, създаването на подходящ модел и изпращането му до изгледа, където да се визуализира. Той трябва винаги да бъде дефиниран с анотация `@Controller`, която указва на Spring да сканира класа за методи с анотации `@RequestMapping`, както и по-кратките варианти `@GetMapping` и `@PostMapping`.

Дефинира се метод с анотацията `@GetMapping`, което е индикация, че дефинирания път в параметъра на анотацията е част от URL, за който приложението ще слуша на този адрес. Също така указва че HTTP метода, по който ще се изпълняват заявката към този адрес ще е GET. String параметъра отговаря за това върху кой URL ще се слуша на дадения метод.

Дефинира се метод с анотацията `@PostMapping`, което е индикация, че дефинирания път в параметъра на анотацията е част от URL, за който приложението ще слуша на този адрес. Също така указва че HTTP метода, по който ще се изпълняват заявката към този адрес ще е POST. String параметъра отговаря за това върху кой URL ще се слуша на дадения метод.

Ако в методите има код, който е възможно да доведе до грешки, е добре да добавим някакъв вид логване. В този случай е добавено логване използвайки анотацията `@Slf4j` от Lombok библиотеката.

При тип на връщане `ModelAndView` от метода, към който е дефиниран `@RequestMapping/@GetMapping/@PostMapping`, се връща модел с данни към изглед частта. В `ModelMap` се добавят всички нужни параметри и обекти от сървърната част и се подават към изгледа, към който ще се визуализират. Като String се въвежда изгледа, който да се търси. В параметрите на метода могат да бъдат дефинирани `@RequestParam` атрибути, които очаква при извикването на конкретния URL.

Анотацията `@ResponseBody` казва на контролера, че съответния метод връща обект, който трябва да бъде поставен в body-то на HTTP отговора.

Върнатият Java обект бива конвертиран в JSON, XML или друг формат подходящ за REST приложения.

Примерна част от контролер за обработка на ролята и статуса на потребителя е показан по-долу:

```
@Slf4j
@Controller
public class HomeController extends BaseController {

    @Autowired
    private final AlgoSettingsService service;

    @PostMapping(View.INDEX_URL)
    public ModelAndView uploadFile(@RequestParam(RequestAttribute.FILE)
        MultipartFile mFile, ModelAndView mav) {
        . . . . .
        return view(View.INDEX_VIEW, mav);
    }

    @GetMapping(View.DOWNLOAD_URL)
    @ResponseBody
    public FileSystemResource downloadFile() {
        . . . . .
        } catch (IOException e) {
            log.error("Couldn't recreate missing file!", e);
        }
    }
    return new FileSystemResource(file);
}
}
```

- View (изглед)

Изгледа представлява визуалната част, която потребителя вижда и взаимодейства с нея. Изгледа отговаря за обработката и визуализацията на информацията от модела и като краен резултат генерира HTML страница, която браузъра интерпретира.

Има няколко основни варианта, чрез които да се дефинира изгледа, като например Thymeleaf и JSP. Thymeleaf е по-новата версия на изгледи чрез Spring. В текущата разработка на системата се използва Thymeleaf.

В изгледа могат да бъдат дефинирани цикли за обхождане на списък от данни, if проверки, различни стойности и данни, както и преизползването на части от страници (layout-и), като например header-и, менюта, footer-и и много други. Командите от Thymeleaf се достъпват чрез префикса “th:” и името на командата.

Основни операции в Thymeleaf:

- th:text – Чрез използването на тази команда се дефинира какъв ще бъде текста, който ще се визуализира в html елемента, където се използва командата.
- th:object – Използва се в html форма, като се указва java обект от модела към форма от изгледа. Данните от формата се запамятват в java обекта, който е дефиниран във елемента от изгледа и изпратен от контролера като модел.
- th:field – Дефинира се във form елемент, като поле от java обект, към което да се свързва стойността от даден елемент в изгледа. Като например текстово поле от изгледа със текстово поле от java обект. При изпращане на формата, стойността се свързва с модела изпратен от контролера.
- th:block – Използва се за дефинирането на блоков елемент с операции от Thymeleaf, като цикъл или if проверка.

- th:if – Проверка, очакваща булев израз.
- th:each – Цикъл в изгледа за обхождане на списъци от данни изпратени от контролер чрез модела

Пример за изглед с използване на фрагменти (изглед отговарящ на начална страница):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head><th:block th:replace="fragments/header"></th:block></head>
<body>
  <div class="custom-shadows" th:include="fragments/navbar"></div>
  <div class="container jumbotron">
    <th:block th:replace="fragments/alert"></th:block>
    . . . .
    <th:block th:replace="fragments/itemset_list(itemset=${comSet},
      titleText=${commonItemSetTitle})"> </th:block>

    <div class="mt-5"
      th:include="fragments/itemset_list(itemset=${rareItemSet},
        titleText=${rareItemSetTitle})"></div>

    <th:block th:replace="fragments/bar_graph"></th:block>
    . . . .
    <th:block th:replace="fragments/footer"></th:block>
    . . . .
</html>
```

Пример за реализиране на фрагмент с параметри и Thymeleaf операции th:each и th:if:

```
<th:block th:fragment="itemset_list (itemset, titleText)">
<div align="center" th:if="${itemset != null and !itemset.empty}">
    <table class="table table-striped">
        . . . .
        <tr th:each="item, iter : ${itemset}">
            . . . .
            <td th:text="${item.value} + '%"></td>
        </tr>
    </table>
</div>
</th:block>
```

- build.gradle

Рова е конфигурационния файл, използван от gradle. В него се описват всички библиотеки, build скриптове, както и обща информация за проекта.

```
plugins {
    id 'java-library'
    . . . .
}

bootJar {
    baseName = 'data-miner'
    version = '0.1.0'
}

sourceCompatibility = 11.0
targetCompatibility = 11.0

repositories {
    jcenter()
    . . . .
}

dependencies {
    compile('org.springframework.boot:spring-boot-starter-thymeleaf')
    . . . .
}
```

- application.properties

Този файл се използва за конфигуриране на многобройни свойства за всякакви библиотеки. В него може да се дефинира връзката на приложението с базата данни, кое ниво грешки да се логват, на кой порт да слуша Tomcat сървър, настройки за Multipart файлове, дали Thymeleaf да има кеширане, дали да има изчакване от страна на DevTools при промяна преди да задейства рестарт на приложението и много други. Пълният лист с възможни конфигурации нараства постоянно и може да се види в документацията на Spring [тук](#).

Примерен application.properties файл:

```
#Server Properties
server.port=8000
spring.thymeleaf.cache=false

# MULTIPART (MultipartProperties)
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=15MB
spring.servlet.multipart.max-request-size=15MB
spring.servlet.multipart.file-size-threshold=0B
server.tomcat.basedir=${user.home}/AppData/Local/Temp/DataMiner8000
spring.servlet.multipart.location=temp_file

#Logging
logging.level.root=WARN
logging.level.com.dataminer=WARN
logging.path = D:/DataMiner/log_folder
logging.file = D:/DataMiner/log_folder/dataminer_log.log

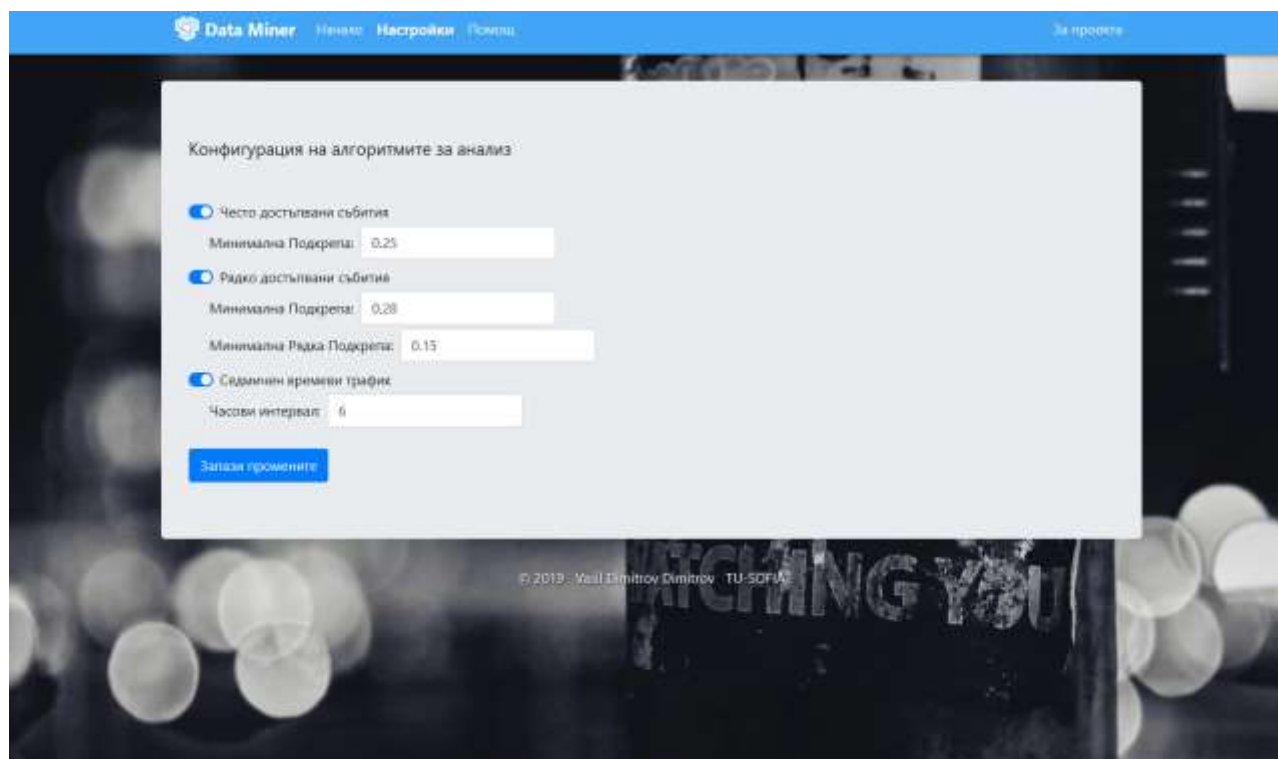
#Database Properties
spring.datasource.driverClassName = com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=password
spring.datasource.url=jdbc:mysql://localhost:3306/data_miner?
spring.jpa.hibernate.ddl-auto=update

#JPA Properties
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.hibernate.format_sql=TRUE
```

4.2. Реализация на потребителския интерфейс

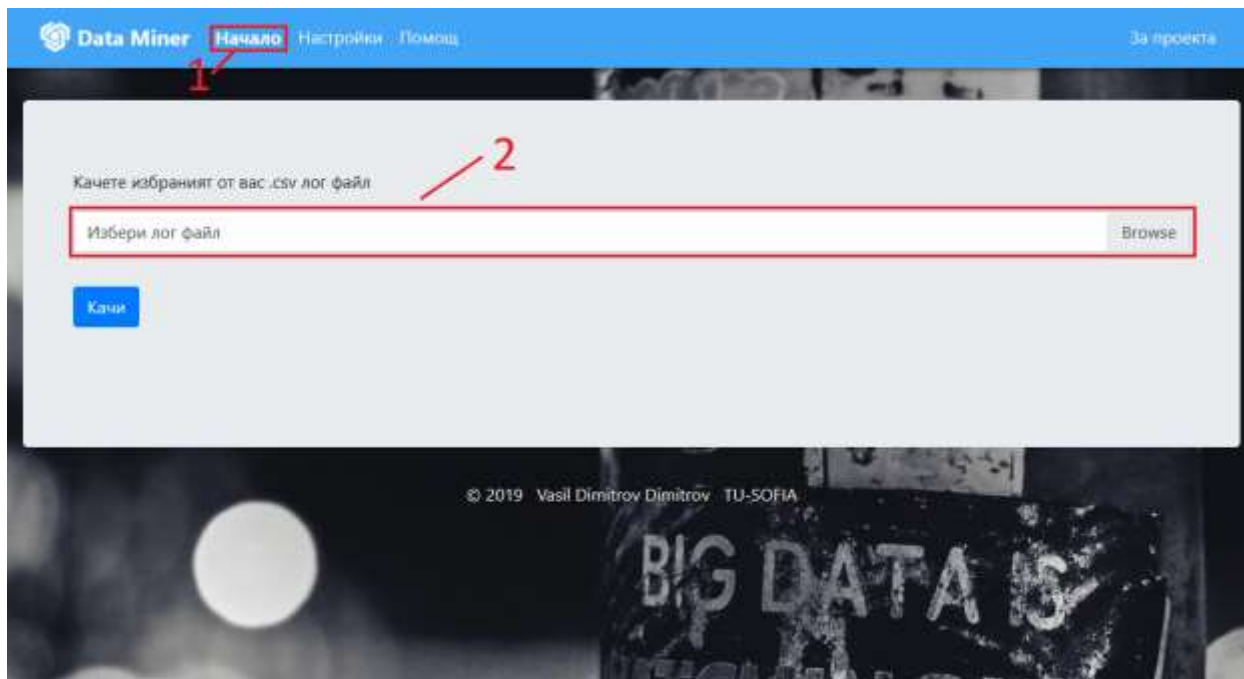
За стилизиране на приложението е избран Bootstrap CCS framework-a. Дизайнът е устроен да е responsive, за да може да се използва от различни устройства. Страниците са устроени с опростен и подреден вид, който да е приветлив за потребителя и нещата да са подредени по начин, по който потребителя очаква да бъдат там. Направени са по такъв начин, че всеки потребител да се ориентира лесно и това което иска да направи да бъде на място, където той иска да бъде. Добавени са сенки на различни елементи за да се подобри четливостта, както и да придадат естествен вид, като на обекти от ежедневието.

Пример:



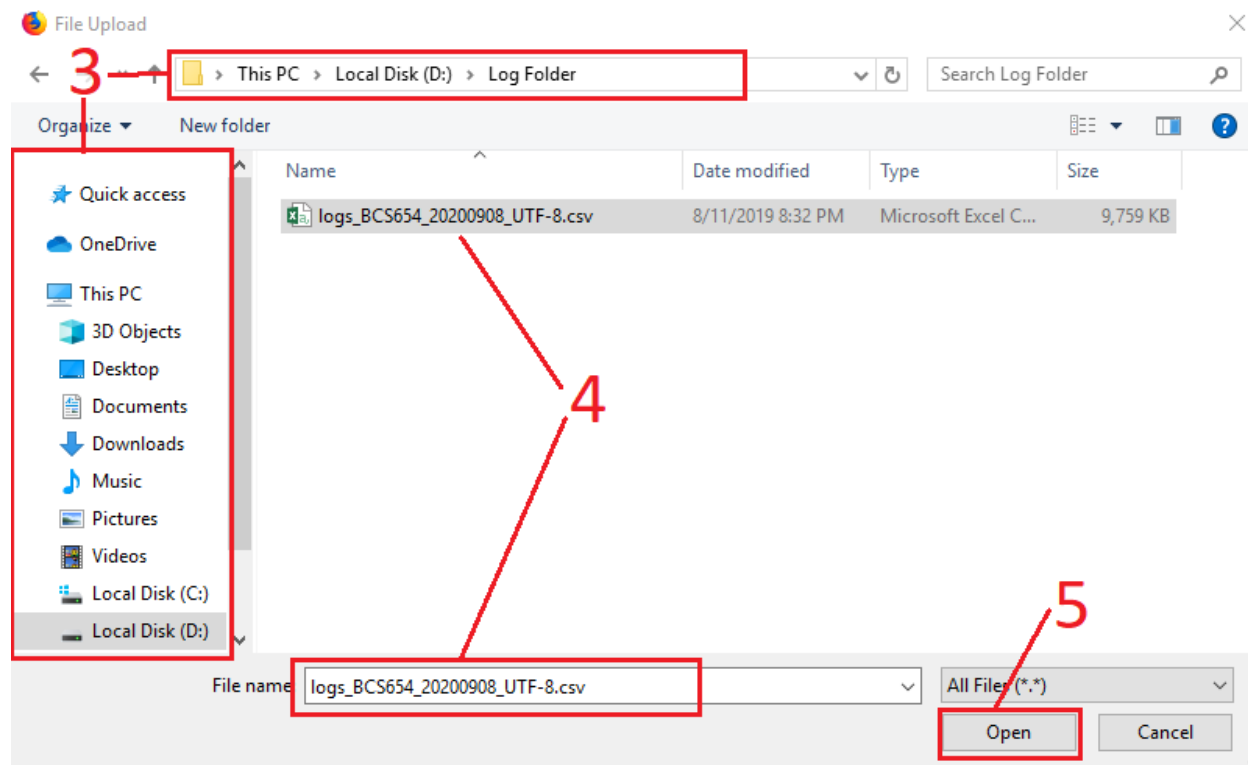
5. Ръководство за работа със системата

5.1. Качване на лог файл



1. Първо трябва да се уверите, че сте на началната страница. Това може да се провери като се провери дали меню таб-ът „Начало“ е удебелен.

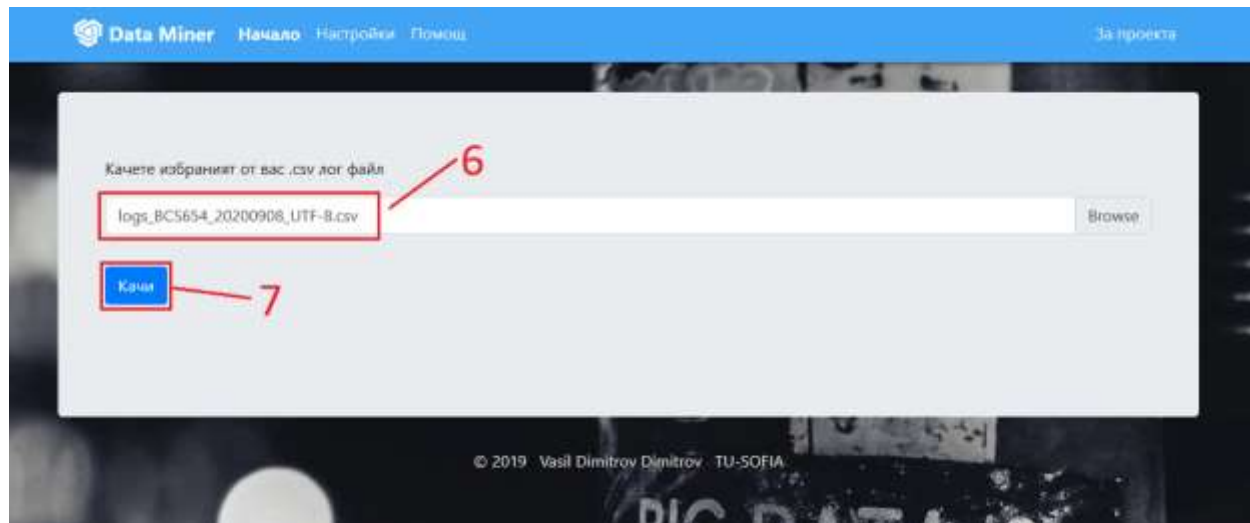
2. След това се кликва върху бутона “Browse” или върху полето за качване на лог файл, обозначено на снимката, при което трябва да се отвори нов прозорец за навигиране в локалните данни на компютъра.



3. Използвайки страничната лява лента или директно адресната лента се навигира до местоположението са лог файлът, който желаем да качим за анализ.

4. Маркира се файлът чрез кликване върху него. За да се уверим, че това е станало успешно може да проверим текстовото поле в дъното на страницата, там трябва да е изписано името на файлът.

5. Кликваме върху бутон Open / Отвори, при което прозорецът ще се затвори и ние ще сме отново на началната страница на сайта.



6. За да се уверим, че ще се качи правилният файл може да проверим дали текста изписан в полето свързано с т. 6 на снимката отговаря на името на файлът, който желаем да качим.

7. Ако всичко изглежда изрядно може да натиснем бутон „Качи“ при което системата ще извърши съответните анализи по лог файлът.

5.2. Преглед на резултати

Ако качването е успешно ще видим съобщение за успешна обработка в началото на страницата. Под панела за качване на файл трябва да се визуализират активните анализи, а под тях бутон за сваляне на подробна информация от анализа:

The screenshot shows the 'Data Miner' web application interface. At the top, there is a blue navigation bar with the 'Data Miner' logo and links for 'Начало', 'Настройки', and 'Помощ'. On the right side of the bar, there is a link 'За проекта'. Below the navigation bar, a green message box states: 'Файл logs_BCS37_20181103_UTF-8-small-bg.csv бе успешно обработен!'. Below this, a section for uploading a file is titled 'Качете избраният от вас .csv лог файл'. It includes a text input field labeled 'Избери лог файл' and a 'Browse' button. A blue 'Качи' button is positioned below the input field. The main content area is titled 'Често достъпвани събития' and contains a table with the following data:

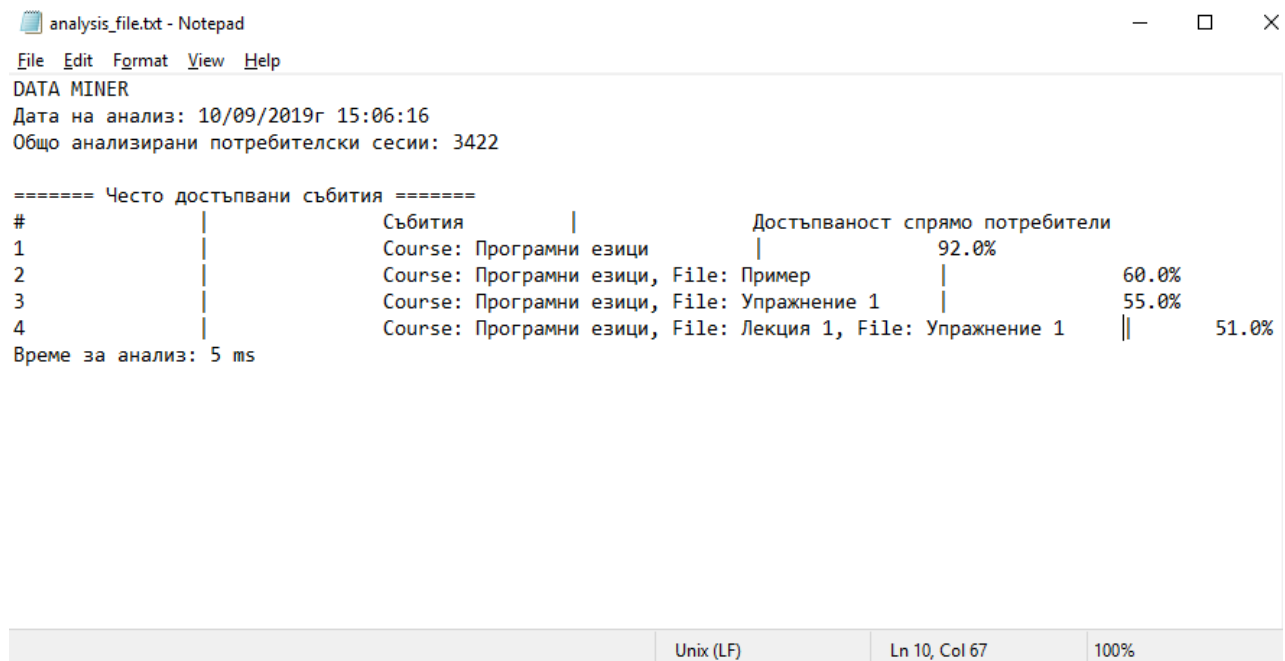
#	Събития	Достъпваност спрямо потребители
1	Course: Програмни езици	92%
2	Course: Програмни езици, File: Пример	60%
3	Course: Програмни езици, File: Упражнение 1	55%
4	Course: Програмни езици, File: Лекция 1, File: Упражнение 1	51%

Below the table, there is a blue button labeled 'Свали подробен анализ'. At the bottom of the page, a footer indicates '© 2019 Vasil Dimitrov Dimitrov TU-SOFIA'.

Ако желаем по-подробна информация или просто да запазим резултатите, може да свалим текстов файл с анализа на компютъра си чрез натискане на бутон „Свали подробен анализ“, който се намира на дъното на страницата, под всички визуализирани резултати. В съдържанието на сваленият файл се съдържа информация за:

- Резултатите от активните анализи
- Дата на извършения анализ
- Общ брой потребителски сесии, който може да се използва за да се изчисли точни брой на потребители достъпвали съответното събитие / комбинация от събитие
- Време за извършване на съответния анализ

Примерен свален файл:



```

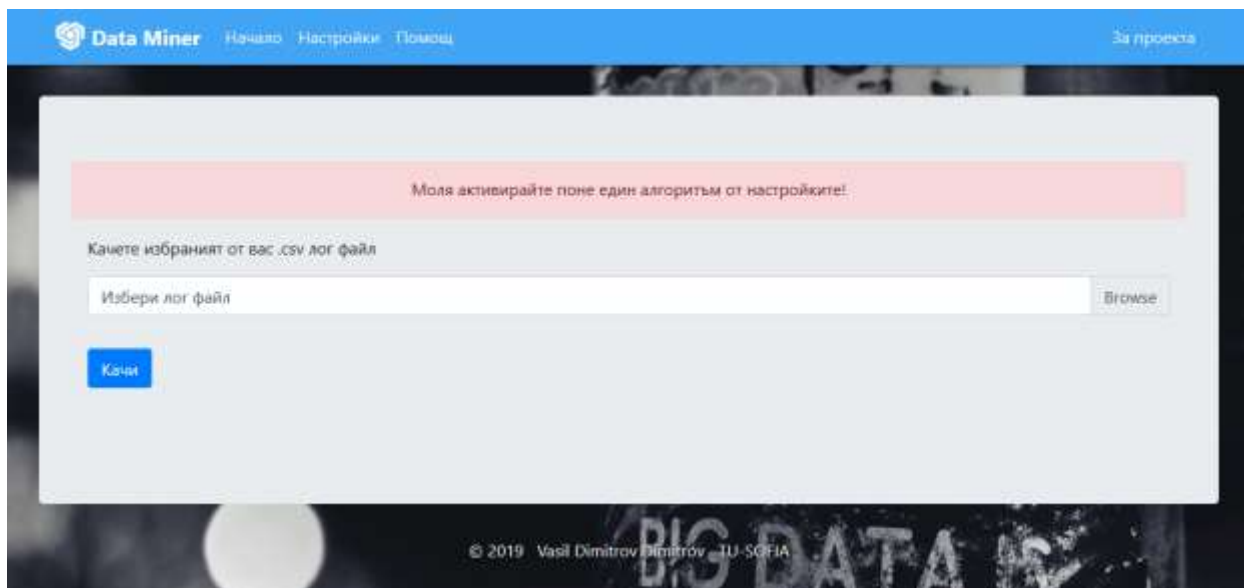
analysis_file.txt - Notepad
File Edit Format View Help
DATA MINER
Дата на анализ: 10/09/2019г 15:06:16
Общо анализирани потребителски сесии: 3422

===== Често достъпвани събития =====
#          |          Събития          |          Достъпваност спрямо потребители
1          |          Course: Програмни езици          |          92.0%
2          |          Course: Програмни езици, File: Пример          |          60.0%
3          |          Course: Програмни езици, File: Упражнение 1          |          55.0%
4          |          Course: Програмни езици, File: Лекция 1, File: Упражнение 1          ||          51.0%
Време за анализ: 5 ms
  
```

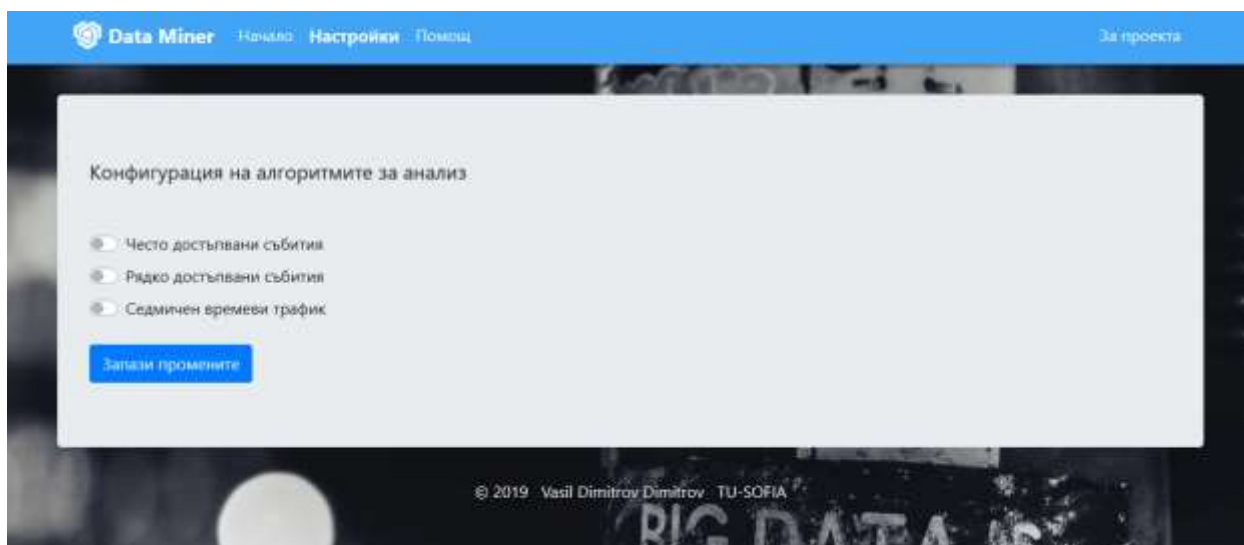
Unix (LF) Ln 10, Col 67 100%

5.3. Конфигурация на алгоритмите за анализ

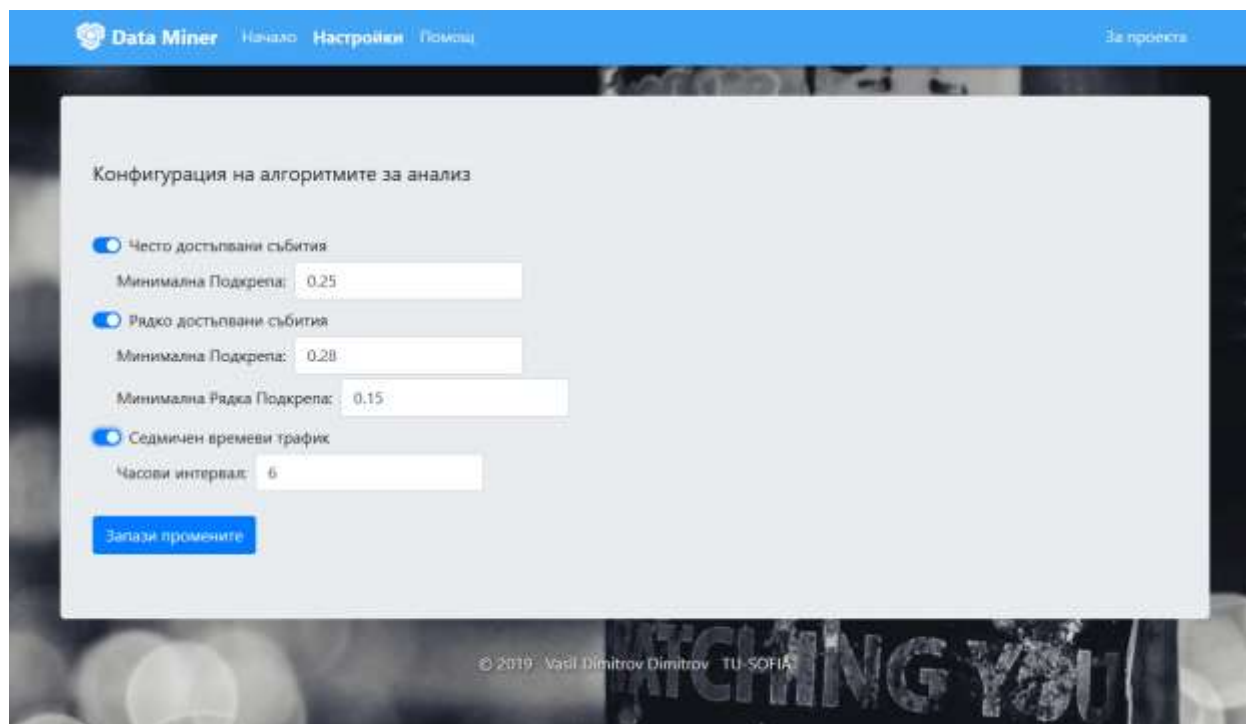
Въпреки успешното следване на стъпките за качване на файл е възможно да получим грешка вместо резултати, която ни подканва да направим някакви промени в настройките на алгоритмите:



Тази грешка се показва, когато всички алгоритми са деактивирани от настройките. Може да се уверим в това чрез посещаване на страницата за настройки чрез натискане на бутона „Настройки“ от менюто на сайта. Ако всички алгоритми са деактивирани трябва страницата да изглежда така:



За да спрем да получаваме гореспоменатата грешка е нужно да активираме поне един от 3те алгоритми. Това става чрез поставяне на отметка или текста на изборния от нас алгоритъм. При активиране ще се оцвети от сиво в синьо и ще се появят допълнителни настройки, позволяващи ни да конфигурираме съответния алгоритъм:



Единствено активните алгоритми ще бъдат визуализирани след качване на файл, като съответните анализи ще бъдат пропуснати изцяло в кода. Това може да се използва за забързване на обработката, ако файлът е твърде голям (над 1 милион реда).

Преди да се премине към конфигурирането на логаритмите трябва да се мине през няколко изрази:

- **Множество от Събития** – списък без повтарящи елементи от потребителски действия.
- **Подкрепа** – това е броят на потребителски сесии достъпили даденото множество от събития. Ако имаме 10 потребителя и 4 от тях са достъпили даденото множество от събития, то се води че това множество е с подкрепа 0,4 или 40%.
- **Чести събития** – Множество от събития, които са били посетени поне или повече от предварително определен минимален брой подкрепа.
- **Затворени чести събития** – Това са „чести събития“, които не присъстват в друго надмножество от чести събития със същата подкрепа.
- **Минимална подкрепа** – това е подкрепата използвана като долна граница за намиране на чести събития и използвана като горна граница за намиране на затворени чести събития.

5.3.1. Често достъпвани събития

Този анализ се извършва чрез алгоритъма “LCM”, който е избран за победител на състезанието FIMI (Frequent Itemset Mining Implementations) през 2004г като най-бързия алгоритъм за откриване на затворени чести множества от събития. Той може да се конфигурира чрез промяна в стойността на:

- **Минимална подкрепа** – това е подкрепата използвана като долна граница за намиране на чести събития. Позволени са стойности от 0 до 1.

5.3.2. Редки събития

Този анализ се извършва чрез алгоритъма “RPGrowth”, който е адаптация на “RP-Tree” алгоритъма за намиране на редки множества от събития с поне едно рядко събитие в транзакционна база данни или лог. Тъй като е адаптиран от “RP-Tree” алгоритъма запазва скоростта, ефективността на паметта и употребата на дърворидна структура. Той работи с 2 параметъра:

- **Минимална подкрепа** – това е подкрепата използвана като горна граница за намиране на затворени чести събития. Позволени са стойности от 0 до 1.
- **Минимална рядка подкрепа** – определя минималната подкрепа за редки събития. Позволени са стойности от 0 до 1. Стойността трябва да е по-малка от тази на минималната подкрепа.

Пример от „Помощ“ страницата на приложението:

Потребител	Достъпвани събития
1	9 10 ...
2	9 10 ...
3	9 ...
4	10 ...
...	...
10	...

Резултат при **Минимална Подкрепа: 4** (събитие трябва да е достъпвано от по-малко от 4 потребители) и **Минимална Рядка Подкрепа: 1** (събитие трябва да е достъпван от 1 или повече потребители):

#	Събития	Брой достъпвания
1	9 10	2
2	9	3
3	10	3

Резултат при **Минимална Подкрепа: 3** и **Минимална Рядка Подкрепа: 1**:

#	Събития	Брой достъпвания
---	---------	------------------

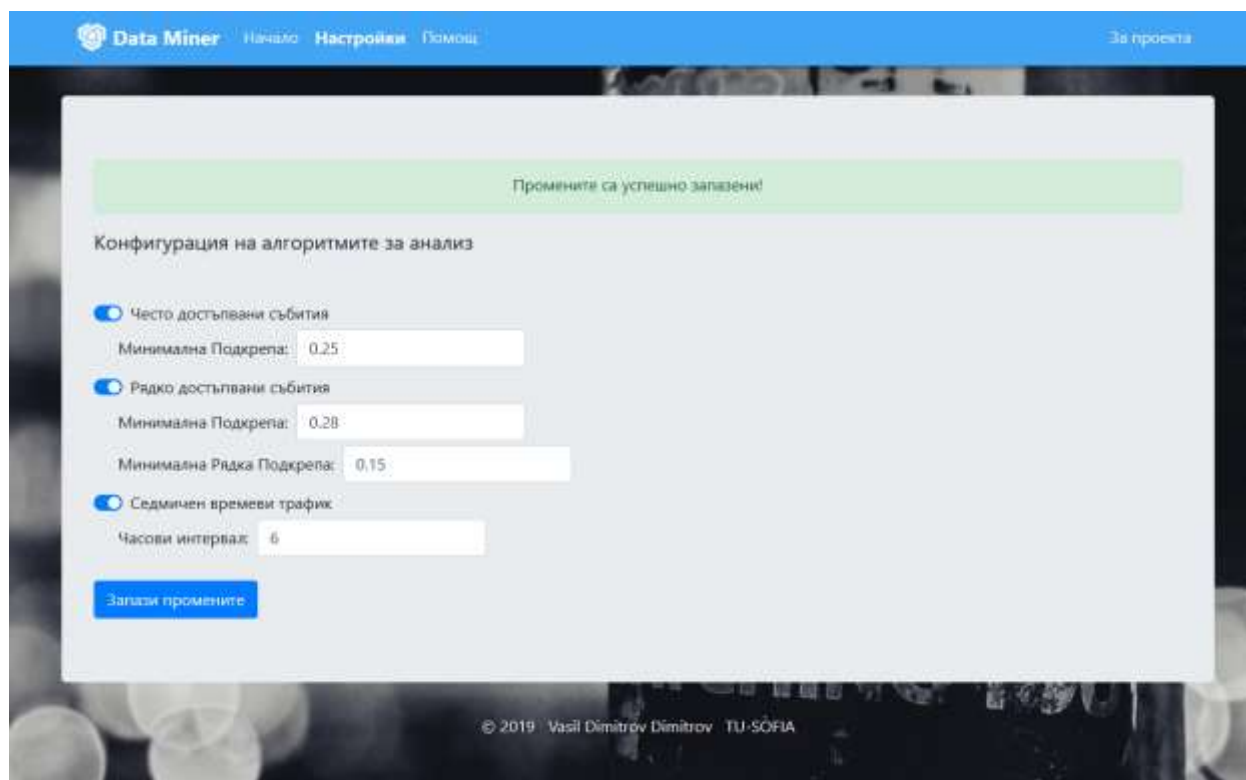
Комбинацията 9 10 с брой достъпвания 2, въпреки че влиза в периода $1 < x < 4$, няма да се върне като резултат, тъй като в себе си не съдържа поне един рядък елемент - събития 9 и 10 поотделно присъстват в 3 на брой достъпвания, което ги прави не рядки.

5.3.3. Седмичен времеви трафик

Този анализ се извършва чрез „VTSA“. Алгоритъмът е имплементиран от автора и служи за показване на най-натоварените и спокойни часове в период от седмица. Върнатите данни могат да се използват за планиране на спиране на системата в най-спокойните часове от седмицата или за поставяне на по-скъпо платени реклами в часовете с повече трафик.

- Часови интервал – часовият интервал на който ще бъде разбит всеки ден от седмицата за анализ. Колкото е по-голяма тази стойност, толкова по-бързо минава анализът. Позволените стойности са от 4 до 24 (включително и за 2те стойности), като е задължително 24 / въведената стойност да връща цяло число.

След като са въведени желаните конфигурации трябва да се натисне бутон „Запази промените“, в противен случай промените няма да се отразят на функционалността на приложението. Ако всички данни са въведени коректно и са приети от базата данни то приложението ще върне съобщение за това:



В случай че не сме въвели някоя стойност коректно системата ще върне смислено съобщение, информиращо ни коя стойност е въведена грешно:

Рядко достъпвани събития: Поле Минимална Рядка Подкрепа трябва да бъде с по-малка стойност от поле Минимална Подкрепа **1**
Часови интервал: Невалидна въведена стойност! Позволените стойности: $4 \leq X \leq 24$ и $24/x$ трябва да дава цяло число. **2**

Конфигурация на алгоритмите за анализ

☒ Често достъпвани събития
Минимална Подкрепа: 0.25

☒ Рядко достъпвани събития
Минимална Подкрепа: 0.28 **1**
Минимална Рядка Подкрепа: 0.5

☒ Седмичен времеви трафик
Часови интервал: 1 **2**

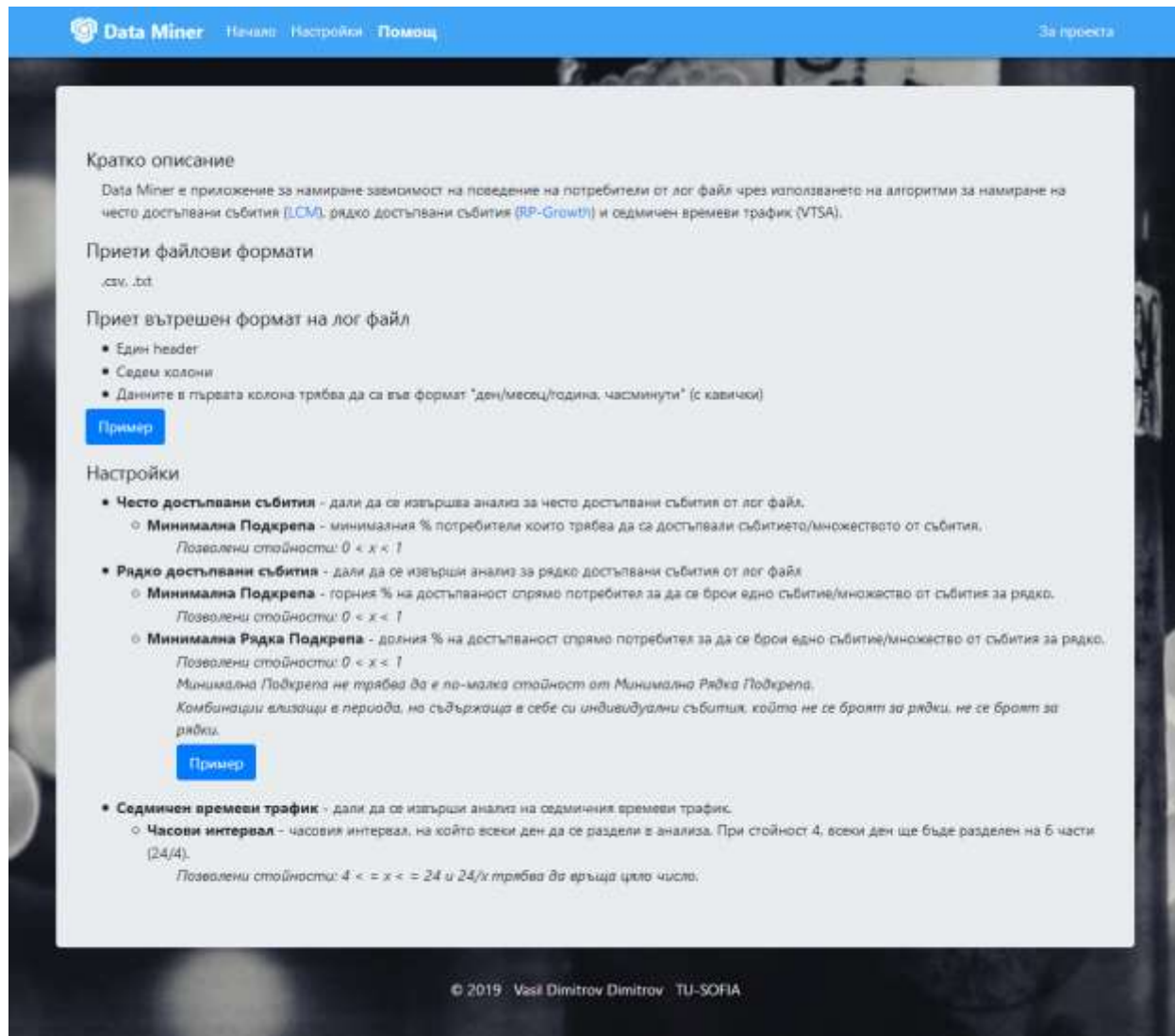
Запази промените

© 2019 Vasil Dimitrov Dimitrov TU-SOFIA

В този случай системата ни информира с 2 различни грешки че минималната подкрепа не трябва да е по-малка стойност от минималната рядка подкрепа за Рядко достъпвани елементи и че стойността за часови интервал не е в позволения диапазон от 4 до 24.

5.4. Достъп до кратко ръководство от сайта

Изградена е страница която е сбито обобщение на ръководството за потребител описано тук, но със споменати само най-важни и съществени части. Страницата е изградена с възможно най-опростени описания и примери за по-сложните обяснения, като също така е структурирана в абзаци и таблици за по-лесно ориентиране от страна на клиента:



Data Miner Начало Настройки Помощ За проекта

Кратко описание

Data Miner е приложение за намиране зависимост на поведение на потребители от лог файл чрез използването на алгоритми за намиране на често достъпвани събития (LCM), рядко достъпвани събития (RP-Growth) и седмичен времеви трафик (VTSA).

Приети файлови формати

.csv, .txt

Приет вътрешен формат на лог файл

- Един header
- Седем колони
- Данните в първата колона трябва да са във формат "ден/месец/година, час:минути" (с кавички)

[Пример](#)

Настройки

- Често достъпвани събития** - дали да се извършва анализ за често достъпвани събития от лог файл.
 - Минимална Подкрепа** - минималния % потребители които трябва да са достъпвали събитията/множеството от събития.
Позволените стойности: $0 < x < 1$
- Рядко достъпвани събития** - дали да се извърши анализ за рядко достъпвани събития от лог файл.
 - Минимална Подкрепа** - горния % на достъпваност спрямо потребителя да да се брои едно събитие/множество от събития за рядко.
Позволените стойности: $0 < x < 1$
 - Минимална Рядка Подкрепа** - долния % на достъпваност спрямо потребителя да да се брои едно събитие/множество от събития за рядко.
Позволените стойности: $0 < x < 1$
Минимална Подкрепа не трябва да е по-малка стойност от Минимална Рядка Подкрепа.
Комбинации влизат в периода, но съдържаща в себе си индивидуални събития, които не се броят за рядки, не се броят за рядки.
- Седмичен времеви трафик** - дали да се извърши анализ на седмичния времеви трафик.
 - Часови интервал** - часовия интервал, на който всеки ден да се раздели в анализа. При стойност 4, всеки ден ще бъде разделен на 6 части (24/4).
Позволените стойности: $4 <= x <= 24$ и $24/x$ трябва да връща цяло число.

[Пример](#)

© 2019 Vasil Dimitrov Dimitrov TU-SOFIA

Добавени са и примери, които с цел лесно разчитане и разбиране са сравнително обемисти. Въпреки че са полезни, ако се добавят в страницата те биха я направили двойно по-голяма, това ще доведе до негативни последици тъй като потребителят ще има затруднения да се ориентира къде да намери правилната информация която търси и коя информация да игнорира. Точно поради тази причина примерите са добавени в modal-и, които се визуализират чрез pop-up window единствено когато потребителя има нужда от тях и кликне съответния бутон:

Потребител **Достъпвани събития**

1	9 10 ...
2	9 10 ...
3	9 ...
4	10 ...
...	...
10	...

Резултат при **Минимална Подкрепа: 4** (събитие трябва да е достъпвано от по-малко от 4 потребители) и **Минимална Рядка Подкрепа: 1** (събитие трябва да е достъпван от 1 или повече потребители):

#	Събития	Брой достъпвания
1	9 10	2
2	9	3
3	10	3

Резултат при **Минимална Подкрепа: 3** и **Минимална Рядка Подкрепа: 1**:

#	Събития	Брой достъпвания
1	9 10	2
2	9	3
3	10	3

Комбинацията 9 10 с брой достъпвания 2, въпреки че влиза в периода $1 < x < 4$, няма да се върне като резултат, тъй като в себе си не съдържа поне един рядък елемент – събития 9 и 10 поотделно присъстват в 3 на брой достъпвания, което ги прави не рядки.

Позволявани стойности: $4 < x < 24$ и $24/x$ трябва да бъде цяло число.

© 2019 Vasil Dimitrov Dimitrov TU-SOFIA

6. Заключение

Потребителите могат да използват системата лесно и интуитивно. Сайтът е лесен за навигиране и ориентация, дори и за нови потребители. Той може да бъде достъпен както от настолен компютър, така и от мобилен телефон, без да се губи подредбата на елементите.

Имплементираните алгоритми са оптимизирани да работят бързо и използвайки малко памет, като връщат полезни анализи за подадените данни. Върнатата информация бива графично представена по четим и лесно разбираем начин, като при нужда от повече подробности има възможност за запазване на текстови файл с пълните резултати от анализите.

Меню помощ има кратка и ясна информация за успешно конфигуриране и работа с функционалностите на приложението.

В резултат на използваните технологии и библиотеки, приложението е в завършен вид, като основата цел е изпълнена и проблема е решен.

7. Литература

7.1. Използвани външни библиотеки:

1. Daniel Fernández: [Thymeleaf](https://www.thymeleaf.org/): <https://www.thymeleaf.org/>
2. Kevin Bourrillion and Jared Levy: [Google/Guava](https://github.com/google/guava), Google: <https://github.com/google/guava>
3. Mark Otto, Jacob Thornton: [Bootstrap](https://getbootstrap.com/): <https://getbootstrap.com/>
4. Perry Nguyen, Reinier Zwitserloot, Roel Spilker, Tor Norbye, Jan Lahoda, and Petr Jirick: [Project Lombok](https://projectlombok.org/), The Project Lombok Authors: <https://projectlombok.org/>
5. [Spring Framework](https://spring.io/), by Pivotal Software: <https://spring.io/>

Всички библиотеки използвани за направата на приложението са с отворен код и безплатен лиценз.

7.2. Използвани онлайн ресурси:

1. Aleksandar Kovachev: [Article](https://github.com/AleksandarKovachev/Article/) - a web-based application for managing the publishing processes in a scientific magazine. GutHub. July 2, 2018: <https://github.com/AleksandarKovachev/Article/>
2. Alphabet Inc: [Google.com](https://www.google.com)
3. Basant kumar Hota: [Spring-HighChart-Graph](https://github.com/Java-Techie-jt/Spring-HighChart-Graph). Java Techie, GitHub. March 29, 2018: <https://github.com/Java-Techie-jt/Spring-HighChart-Graph>
4. Daniel Morales: [Uploader](https://github.com/danielm/uploader). GitHub. January 16, 2018: <https://github.com/danielm/uploader>
5. Eugen: [Baeldung](https://www.baeldung.com/): <https://www.baeldung.com/>
6. Eugen Hoble: [Dzone.com - Reading CSV file](https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream) by Integration Zone, September 28, 2016: <https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream>

7. Francis: [NHSystem](#). GitHub. January 11, 2018:
<https://github.com/sambaf/NHSystem>
8. Greg Turnquist: [ags-uploading-files](#). Spring Guides, GitHub. August 12, 2019: <https://github.com/spring-guides/gs-uploading-files>
9. Guava's [BiMap](#):
<https://github.com/google/guava/wiki/NewCollectionTypesExplained>
10. Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao: [Mining Frequent Patterns without Candidate Generation](#): A Frequent-Pattern Tree Approach. Data Min. Knowl. Discov. 8(1): 53-87 (2004):
<https://www.cs.sfu.ca/~jpei/publications/sigmod00.pdf>
11. Mark Otto, Jacob Thornton: [GetBootstrap - documentation](#):
<https://getbootstrap.com/docs/4.0/>
12. Philippe Fournier Viger: [SPMF](#) and [PPSF](#), Harbin Institute of Technology (Shenzhen): <http://www.philippe-fournier-viger.com/spmf/>, <http://ppsf.ikelab.net/>
13. Refsnes Data: [W3Schools.com](#)
14. Sidney Tsang, Yun Sing Koh, Gillian Dobbie, [RP-Tree: Rare Pattern Tree Mining](#), International Conference of Data Warehousing and Knowledge Discovery, 277-288 (2011):
https://link.springer.com/chapter/10.1007/978-3-642-23544-3_21
15. [StackOverflow.com](#)
16. Takeaki Uno, Masashi Kiyomi and Hiroki Arimura (2004). [LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets](#). Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations Brighton, UK, November 1, 2004:
<http://www.philippe-fournier-viger.com/spmf/LCM2.pdf>
17. Daniel Fernández: [Thymeleaf - documentation](#):
<https://www.thymeleaf.org/documentation.html>

18. Umesh Awasthi: [JavaDevJournal.com - Java 8 – How to Format LocalDateTime](https://www.javadevjournal.com/java/java-8-format-localdatetime) By Java Development Journal, November 30, 2018:
<https://www.javadevjournal.com/java/java-8-format-localdatetime>
19. [Unsplash.com](https://unsplash.com/): <https://unsplash.com/>
20. [Wikipedia - Извличане на знания от данни](https://bg.wikipedia.org/wiki/%D0%98%D0%B7%D0%B2%D0%BB%D0%B8%D1%87%D0%B0%D0%BD%D0%B5_%D0%BD%D0%B0_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BE%D1%82_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8) :
https://bg.wikipedia.org/wiki/%D0%98%D0%B7%D0%B2%D0%BB%D0%B8%D1%87%D0%B0%D0%BD%D0%B5_%D0%BD%D0%B0_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BE%D1%82_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8
21. Yong Mook Kim: [Mkyong.com](https://mkyong.com)

8. Приложение

Част от сорс кода:

```
/**
 * This class is meant to save some repeating code
 * across the different controllers
 *
 * @author Vasil.Dimitrov^2
 */
public abstract class BaseController {
    public ModelAndView view(String viewName) {
        return this.view(viewName, new ModelAndView());
    }

    /**
     * Adds the correct prefix to the viewName, then
     * adds the viewName itself to the model.
     * Finally returns the {@link ModelAndView}
     *
     * @param viewName
     * @param model
     * @return
     */
    public ModelAndView view(String viewName, ModelAndView model) {
        model.setViewName(View.VIEWS_PREFIX + viewName);

        return model;
    }

    /**
     * Method meant for redirecting by passing the redirectUrl
     * URL directly without having to add
     * the "redirect:" prefix or create a {@link ModelAndView}
     *
     * @param redirectUrl
     * @return
     */
    public ModelAndView redirect(String redirectUrl) {
        ModelAndView model = new ModelAndView();
        model.setViewName("redirect:" + redirectUrl);

        return model;
    }
}
```

```
/**
 * Controller for working with
 * the /about and the /resources context
 *
 * @author Vasil.Dimitrov^2
 *
 */
@Controller
public class AboutController extends BaseController {

    /**
     * Method returning the /about page with general
     * info about the project and the author.
     *
     * @param modelAndView
     * @return
     */
    @GetMapping(View.ABOUT_URL)
    public ModelAndView showAboutPage(ModelAndView modelAndView) {
        return view(View.ABOUT_VIEW);
    }

    /**
     * Method returning the /resources page with info
     * about the resources used in constructing
     * this application
     *
     * @param modelAndView
     * @return
     */
    @GetMapping(View.RESOURCES_URL)
    public ModelAndView showResourcesPage(ModelAndView modelAndView) {
        return view(View.RESOURCES_VIEW, modelAndView);
    }
}

/**
 * Controller for working with the /help context
 *
 * @author Vasil.Dimitrov^2
 *
 */
@Controller
public class HelpController extends BaseController {

    /**
     * Method returning the /help view
     *
     * @param modelAndView
     * @return
     */
}
```

```

        */
        @GetMapping(View.HELP_URL)
        public ModelAndView showHelpPage(ModelAndView modelAndView) {
            return view(View.HELP_VIEW);
        }
    }

/**
 * Controller for working with the /settings context
 *
 * @author Vasil.Dimitrov^2
 */
@Controller
public class SettingsController extends BaseController {

    private final AlgoSettingsService service;

    @Autowired
    public SettingsController(AlgoSettingsService service) {
        this.service = service;
    }

    /**
     * Method returning the settings view
     *
     * @param modelAndView
     * @return
     */
    @GetMapping(View.SETTINGS_URL)
    public ModelAndView showSettingsPage(ModelAndView modelAndView) {
        modelAndView.addObject(RequestAttribute.ALGO_SETTINGS,
            this.service.getDefaultAlgoSettings());
        return view(View.SETTINGS_VIEW, modelAndView);
    }

    /**
     * Method for saving the new settings page.
     * Before saving, the values are checked if they
     * are valid and an appropriate error message
     * is returned for the user to see.
     *
     * @param algoSettings
     * @param modelAndView
     * @return
     */
    @PostMapping(View.SETTINGS_URL)
    public ModelAndView updateSettingsPage(
        @ModelAttribute(value = RequestAttribute.ALGO_SETTINGS)
        AlgoSettings algoSettings,
        ModelAndView modelAndView) {
        List<String> errors = algoSettings.isValid();
    }

```

```

        if (!CollectionUtils.isEmpty(errors)) {
            modelAndView.addObject(RequestAttribute.ERROR_MSG, errors);
            return view(View.SETTINGS_VIEW, modelAndView);
        }

        if (this.service.saveAlgoSettings(algoSettings)) {
            modelAndView.addObject(RequestAttribute.SUCCESS_MSG,
                "Промените са успешно запазени!");
        } else {
            modelAndView.addObject(RequestAttribute.ERROR_MSG,
                "Възникна грешка при опит за запазване на
промените!");
        }
        System.out.println(algoSettings.toString());
        return view(View.SETTINGS_VIEW, modelAndView);
    }
}

/**
 * Home controller for working with the /index context
 *
 * @author Vasil.Dimitrov^2
 *
 */
@Slf4j
@Controller
public class HomeController extends BaseController {

    private final AlgoSettingsService service;

    @Autowired
    public HomeController(AlgoSettingsService service) {
        this.service = service;
    }

    @GetMapping({ "/", View.INDEX_URL })
    public ModelAndView showIndexPage(ModelAndView modelAndView) {
        return view(View.INDEX_VIEW, modelAndView);
    }

    /**
     * Method handling the log file upload to
     * server as well as the analysis made by the
     * different algorithms
     *
     * @param mFile
     * @param mav
     * @return
     */
    @PostMapping(View.INDEX_URL)
    public ModelAndView uploadFile(
        @RequestParam(RequestAttribute.FILE) MultipartFile mFile,

```

```

        ModelAndView mav) {
    AlgoSettings algoSettings = this.service.getDefaultAlgoSettings();
    List<String> results = new ArrayList<>();

    if (algoSettings.areAllAlgorithmsDisabled()) {
        mav.addObject(RequestAttribute.ERROR_MSG,
            "Моля активирайте поне един алгоритъм от
настройките!");
        return view(View.INDEX_VIEW, mav);
    }

    LogFile logFile = LogFile.createFromMultipartFile(mFile,
algoSettings.getVtSa());
    int sessionsCount = logFile.getUserSessionList().size();

    if (algoSettings.getLcm()) {
        Dataset dataset = new Dataset(logFile);
        Itemsets itemsets = new
AlgoLCM().runAlgorithm(algoSettings.getLcmMinSup(), dataset);
        mav.addObject(RequestAttribute.LCM_DATA,
itemsets.getRelativeItemsets(sessionsCount));
        mav.addObject(RequestAttribute.LCM_TITLE,
Constant.LCM_TITLE_TEXT);
        results.add(itemsets.toString());
    }
    if (algoSettings.getRpg()) {
        Itemsets itemsets = new
AlgoRPGrowth().runAlgorithm(logFile,
                                algoSettings.getRpgMinSup(),
algoSettings.getRpgMinRareSup());
        mav.addObject(RequestAttribute.RPG_DATA,
itemsets.getRelativeItemsets(sessionsCount));
        mav.addObject(RequestAttribute.RPG_TITLE,
Constant.RPG_TITLE_TEXT);
        results.add(itemsets.toString());
    }
    if (algoSettings.getVtSa()) {
        AlgoVTSA algo = new AlgoVTSA();
        algo.run(logFile);
        mav.addObject(RequestAttribute.VTSA_DATA,
algo.getResults());
        results.add(algo.toString());
    }

    HelperUtil.generateFile(results);

    mav.addObject(RequestAttribute.SUCCESS_MSG, "Файл " +
mFile.getOriginalFilename() + " бе успешно обработен!");
    return view(View.INDEX_VIEW, mav);
}

```

```

/**
 * Method returning analysis_file.txt file
 * with more detailed algorithm analysis made on
 * the uploaded log file
 *
 * @return
 */
@GetMapping(View.DOWNLOAD_URL)
@ResponseBody
public FileSystemResource downloadFile() {
    File file = new File(Constant.FILE_UPLOAD_DIR);
    if (!file.exists()) {
        try {
            file.createNewFile();
            FileWriter fw = new FileWriter(file, false);
            fw.write("Файлът не бе намерен");
            fw.close();
            Log.warn("Client tried retrieving file with analysis, but
it was missing!");
        } catch (IOException e) {
            Log.error("Exception occurred trying to recreate a
misssing analysis file!", e);
        }
    }
    return new FileSystemResource(new File(Constant.FILE_UPLOAD_DIR));
}

/**
 * Pojo class for storing the log data in an ordered and easy to analyze structure.
 *
 * @author Vasil.Dimitrov^2
 */
@Getter
@Setter
@Slf4j
public class LogFile {
    private List<UserSession> userSessionList = new ArrayList<>();
    private BiMap<Integer, String> uniqueECMap = HashBiMap.create();
    private Map<Integer, Integer> keyCount = new HashMap<>();

    /**
     * Method for adding a log line to the {@link LogFile} object
     *
     * @param line
     */
    public void addLine(String line, boolean isVtsaOn) {
        String elements[] = line.split("\t");
        Integer lastKey = null;
    }
}

```

```

        if (elements.length != Constant.LOG_FILE_VALID_SIZE) {
            return;
        }

        RawLineElements rawLine = new RawLineElements(elements);

        // adding to the uniqueEventContextMap
        if (!this.uniqueEMap.containsValue(rawLine.getEventContext())) {
            lastKey = this.uniqueEMap.size() + 1;
            this.uniqueEMap.put((lastKey), rawLine.getEventContext());
            this.keyCount.put(lastKey, 1);
        } else {
            lastKey =
this.uniqueEMap.inverse().get(rawLine.getEventContext());
            this.keyCount.put(lastKey, this.keyCount.get(lastKey) + 1);
        }

        UserSession newUserSession;
        try {
            newUserSession = new UserSession(rawLine.getIp(),
                                                isVtsaOn ?
LocalDateTime.parse(rawLine.getDateTime(), Constant.LOG_FILE_DATE_TIME_FORMAT) :
null,
                                                lastKey != null ? lastKey :
this.uniqueEMap.inverse().get(rawLine.getEventContext()));
        } catch (DateTimeParseException e) {
            Log.error("DateTimeParseException was thrown for " +
rawLine.getDateTime(), e);
            return;
        }

        if (this.userSessionList.contains(newUserSession)) {
            int userSessionIndex =
this.userSessionList.indexOf(newUserSession);

            this.userSessionList.get(userSessionIndex).addUserSessionEvent(newUserSession)
;
        } else {
            this.userSessionList.add(newUserSession);
        }
    }

    /**
     * Create a LogFile out of a MultipartFile. Supports Cyrillic encoding.
     *
     * @param mFile
     * @return
     */
    public static LogFile createFromMultipartFile(MultipartFile mFile, boolean
isVtsaOn) {

        LogFile logFile = new LogFile();
    }

```



```

        try (BufferedReader br = new BufferedReader(new
InputStreamReader(mFile.getInputStream(), StandardCharsets.ISO_8859_1))) {
            br.readLine();
            String line;
            while ((line = br.readLine()) != null) {
                logFile.addLine(new String(line.getBytes("Cp1252"),
"Cp1251"), isVtsaOn);
            }
        } catch (IOException e) {
            Log.error("Error creating LogFile from file " +
mFile.getOriginalFilename(), e);
        }

        return logFile;
    }

    /*
     * This toString has been implemented with a limiter as the object itself is
too big
     */
    @Override
    public String toString() {
        int limiter = 1;
        StringBuffer strbf = new StringBuffer();
        for (UserSession us : this.userSessionList) {
            strbf.append("Events for user with ip: " + us.getIp() + "\n");
            int i = 0;
            for (Integer eventContextKey : us.getEventContextKeySet()) {
                strbf.append("\nTime executed: " +
us.getDateTimeList().get(i++));
                strbf.append("\nEvent context: " +
this.uniqueECMap.get(eventContextKey));
                strbf.append("\n");
            }

            if (limiter-- <= 0) {
                break;
            }
        }
        return strbf.toString();
    }
}

/**
 * Intermediate class used for storing and converting the data from the uploaded log
file to the {@link LogFile} object line by line
 *
 * @author Vasil.Dimitrov^2
 *
 */

```

```

@Getter
@Setter
public class RawLineElements {
    private String dateTime;
    private String eventContext;
    private String component;
    private String eventName;
    private String description;
    private String origin;
    private String ip;

    public RawLineElements(String[] elements) {
        if (elements.length == Constant.LOG_FILE_VALID_SIZE) {
            this.dateTime = elements[Constant.LOG_FILE_DATE_TIME_POSITION];
            this.eventContext =
elements[Constant.LOG_FILE_EVENT_CONTEXT_POSITION];
            this.component = elements[Constant.LOG_FILE_COMPONENT_POSITION];
            this.eventName = elements[Constant.LOG_FILE_EVENT_NAME_POSITION];
            this.description =
elements[Constant.LOG_FILE_DESCRIPTION_POSITION];
            this.origin = elements[Constant.LOG_FILE_ORIGIN_POSITION];
            this.ip = elements[Constant.LOG_FILE_IP_POSITION];
        }
    }
}

```

```

/**
 * This class hold a list of a list of itemsets. Where the first level of the list
 * corresponds to the size of the itemsets contained inside it.
 *
 * @author Vasil.Dimitrov^2
 *
 */
public class Itemsets{
    private final List<List<Itemset>> levels = new ArrayList<>();
    @Getter
    private int itemsetsCount = 0;
    private String name;

    public Itemsets(String name) {
        this.name = name;
        // adding empty level 0 by default
        this.levels.add(new ArrayList<Itemset>());
    }

    public void printItemsets() {
        System.out.println(" ----- " + this.name + " -----");
        int patternCount = 0;
        int levelCount = 0;
    }
}

```

```

of items)    // for each level (a level is a set of itemsets having the same number
              for (List<Itemset> level : this.levels) {
                // print how many items are contained in this level
                System.out.println("  L" + levelCount + " ");
                // for each itemset
                for (Itemset itemset : level) {
                  // Arrays.sort(itemset.getItems());
                  // print the itemset
                  System.out.print("    pattern " + patternCount + ": ");
                  itemset.print();
                  // print the support of this itemset
                  System.out.print("support : " +
itemset.getAbsoluteSupport());
                  patternCount++;
                  System.out.println("");
                }
                levelCount++;
              }
              System.out.println(" -----");
            }

    public Map<int[], Double> getRelativeItemsets(int totalTransactionCount) {
        Map<int[], Double> map = new LinkedHashMap<>();
        for (List<Itemset> level : this.levels) {
            for (Itemset itemset : level) {
                double percentage = (double) (100 *
itemset.getAbsoluteSupport() / totalTransactionCount;
                map.put(itemset.getItemset(), HelperUtil.round(percentage,
2));
            }
        }
        return map;
    }

    /**
     * Sort the itemsets on the second level. The sort order is chosen based on
the passed
     * argument.
     *
     * @param reverseOrder
     */
    public void sortItemsets(boolean reverseOrder) {
        if (reverseOrder) {
            for (List<Itemset> level : this.levels) {
                Collections.sort(level, Collections.reverseOrder());
            }
        } else {
            for (List<Itemset> level : this.levels) {
                Collections.sort(level);
            }
        }
    }
}

```

```

    public void addItemset(Itemset itemset, int k) {
        while (this.levels.size() <= k) {
            this.levels.add(new ArrayList<Itemset>());
        }
        this.levels.get(k).add(itemset);
        this.itemsetsCount++;
    }

    public void decreaseItemsetCount() {
        this.itemsetsCount--;
    }
}

/**
 * This class represents an itemset (a set of items) implemented as an array of
 * integers with
 * a variable to store the support count of the itemset.
 *
 * @author Vasil.Dimitrov^2
 */
@Getter
public class Itemset extends AbstractOrderedItemset implements Comparable<Itemset> {
    private int[] itemset;
    @Setter
    private int support = 0;

    /**
     * Constructor used by {@link AlgoRPGrowth}
     *
     * @param items
     */
    public Itemset(int[] items) {
        this.itemset = items;
    }

    /**
     * Constructor used by {@link AlgoLCM}
     *
     * @param itemset
     * @param support
     */
    public Itemset(List<Integer> itemset, int support) {
        this.itemset = new int[itemset.size()];
        int i = 0;
        for (Integer item : itemset) {
            this.itemset[i++] = item.intValue();
        }
        this.support = support;
    }
}

```

```

    }

    public void increaseTransactionCount() {
        this.support++;
    }

    @Override
    public int getAbsoluteSupport() {
        return this.support;
    }

    @Override
    public int size() {
        return this.itemset.length;
    }

    @Override
    public Integer get(int position) {
        return this.itemset[position];
    }

    @Override
    public int hashCode() {
        return Arrays.hashCode(this.itemset);
    }

    @Override
    public int compareTo(Itemset o) {
        int x = this.support;
        int y = o.getAbsoluteSupport();

        return (x < y) ? -1 : ((x == y) ? 0 : 1);
    }
}

<th:block th:fragment="bar_graph">
<div th:if="{surveyMap != null and !surveyMap.empty}">
    <div class="container">
        <h2 align="center">Времеви трафик</h2>
        <div id="container" style="width: 1000px; height: 400px; margin: 0
auto"></div>
    </div>
    <script src="/js/highcharts.js"></script>
    <script src="/js/highcharts.exporting.js"></script>

    <script th:inline="javascript">
$(function(){
    Highcharts.chart('container', {
        chart: {
            type: 'column'

```

```

    },
    title: {
      text: 'Времеви График'
    },
    subtitle: {
      text: ''
    },
    xAxis: {
      categories: [[${surveyMap.keySet()}]],
      crosshair: true
    },
    yAxis: {
      min: 0,
      max: [[${surveyMapMaxValue} + 1]],
      title: {
        text: 'Натовареност на системата [среден брой събития]'
      }
    },
    tooltip: {
      headerFormat: '<span style="font-size:15px">{point.key}</span><table>',
      pointFormat: '<tr><td
style="color:{series.color};padding:0">{series.name}: </td>' +
      '<td style="padding:0"><b>{point.y:.1f}</b> събития</td></tr>',
      footerFormat: '</table>',
      shared: true,
      useHTML: true
    },
    plotOptions: {
      column: {
        pointPadding: 0,
        borderWidth: 0
      }
    },
    series: [{
      name: 'Трафик',
      data: [[${surveyMap.values()}]]
    }]
  });
});
</script>
</div>
</th:block>

```

```
<th:block th:fragment="navbar">
```

```

  <nav class="navbar navbar-expand-md navbar-dark bg-primary-2 custom-navbar"
id="navbar">
    <div class="container">
      <a class="navbar-brand font-weight-bold" href="#">

```

```

        
        Data Miner
    </a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="collapsibleNavbar">
        <ul class="navbar-nav">
            <li class="nav-item"
th:classappend="${#httpServletRequest.getRequestURI() == '/' ? 'active':''}">
                <a class="nav-link text-light" href="/">Начало <span
class="sr-only">(current)</span></a>
            </li>
            <li class="nav-item"
th:classappend="${#httpServletRequest.getRequestURI() == '/settings' ? 'active':''}">
                <a class="nav-link text-light"
href="/settings">Настройки</a>
            </li>
            <li class="nav-item"
th:classappend="${#httpServletRequest.getRequestURI() == '/help' ? 'active':''}">
                <a class="nav-link text-light" href="/help">Помощ</a>
            </li>
        </ul>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item"
th:classappend="${#httpServletRequest.getRequestURI() == '/about' ? 'active':''}">
                <a class="nav-link text-light" href="/about">За
проекта</a>
            </li>
        </ul>
    </div>
</div>
</nav>

<script>
window.onscroll = function() {myFunction()};

var navbar = document.getElementById("navbar");
var sticky = navbar.offsetTop;

function myFunction() {
    if (window.pageYOffset >= sticky) {
        navbar.classList.add("sticky")
    } else {
        navbar.classList.remove("sticky");
    }
}
</script>
</th:block>

```

Целия код може да бъде намерен в GitHub на адреса:

<https://github.com/Vasil-Dimitrov/DataMiner>