

Дипломна работа

Тема:

Стеганографски софтуер за скриване на данни в PNG изображение

Дата: 22.06.2012г

гр. София

Разработил:

Кирил Стефанов
Александров

гр.63, ФКСУ, КСТ

фак. номер: 121207102

Съдържание

1.	Въведение.....	5
1.1.	Какво е стеганография?	5
1.2.	Исторически примери за използването на стеганография	5
2.	Съществуващи приложения	7
2.1.	VSL – Virtual Steganography Laboratory	7
2.1.1.	Кратко описание.....	7
2.1.2.	Поддържани файлови формати.....	7
2.1.3.	Изображения	7
2.1.4.	Добри страни	8
2.1.5.	Лоши страни.....	8
2.2.	Open Stego.....	8
2.2.1.	Кратко описание.....	8
2.2.2.	Поддържани файлови формати.....	9
2.2.3.	Изображения	9
2.2.4.	Добри страни	10
2.2.5.	Лоши страни.....	10
2.3.	Stream Steganography.....	11
2.3.1.	Кратко описание.....	11
2.3.2.	Поддържани файлови формати.....	11
2.3.3.	Добри страни	11
2.3.4.	Лоши страни.....	11
2.4.	Steg Hide	11
2.4.1.	Кратко описание.....	11
2.4.2.	Поддържани файлови формати.....	12
2.4.3.	Изображения	12

2.4.4.	Външни зависимости	12
2.4.5.	Добри страни	12
2.4.6.	Лоши страни.....	12
3.	Описание на използваните среди и технологии	13
3.1.	Операционна система.....	13
3.2.	Език за програмиране	13
3.3.	Интернет сървър.....	14
3.4.	Технологии, използвани при клиента	15
3.5.	Външни зависимости	16
3.5.1.	На сървъра	16
3.5.2.	При клиента	16
4.	Избор на алгоритъм	17
4.1.	Описание на алгоритъма	17
4.1.1.	Побитови операции	17
4.1.2.	Последователност на алгоритъма	19
4.1.3.	Подобрения	22
4.1.4.	Обратната посока	23
5.	Реализация на стеганографския софтуер.....	25
5.1.	Избор на решение.....	25
5.2.	Класова йерархия.....	25
5.3.	Описание на основните класове.....	26
5.3.1.	Steganography	26
5.3.2.	ImageType.....	28
5.3.3.	SteganographyException	28
5.3.4.	ServletResponse.....	28
5.3.5.	EncodeServletResponse	28
5.3.6.	DecodeServletResponse	28

5.3.7.	Message	28
5.3.8.	Status	29
5.3.9.	EncodeImageServlet	29
5.3.10.	DecodeImageServlet	29
5.3.11.	ImageServlet	30
5.3.12.	Constants	30
5.3.13.	ImageUtils	30
5.3.14.	MessageConstants	31
5.3.15.	Utils	31
5.4.	Описание на работата на приложението	31
6.	Тестове	33
6.1.	Тест №1	33
6.2.	Тест №2	36
6.3.	Тест №3	37
6.4.	Тест №4	39
6.5.	Тест №5	41
7.	Изводи	44
8.	Ръководство за потребителя	45
8.1.	Начин на инсталация	45
8.2.	Начин на употреба	45
9.	Заключение	49
10.	Програмен код	50
10.1.	Steganography.java	50
10.2.	ImageType.java	54
10.3.	SteganographyException.java	54
10.4.	DecodeServletResponse.java	55
10.5.	EncodeServletResponse.java	56

10.6.	Message.java.....	56
10.7.	ServletResponse.java	57
10.8.	Status.java	57
10.9.	DecodeImageServlet.java	59
10.10.	EncodeImageServlet.java	61
10.11.	ImageServlet.java.....	64
10.12.	Constants.java	65
10.13.	ImageUtils.java	67
10.14.	Log.java	69
10.15.	MessageConstants.java.....	70
10.16.	Utils.java.....	71
10.17.	Index.jsp	74
10.18.	Utils.js.....	80
10.19.	Style.css	83

1. Въведение

1.1.Какво е стеганография?

Стеганография - това е наука, занимаваща се със скриване на информационно съобщение по такъв начин в информационен обект, че никой друг, освен създателя му да не подозира за неговото съществуване. Стеганография произлиза от гръцките думи *steganos* (στεγανός), което означава "скрит" или "покрит" и *graphei* (γραφῆ), което означава "писменост".

На пръв поглед стеганографията много напомня на криптографията, но всъщност сами по себе си двете науки са доста различни. Криптографията е фокусирана върху обработката на информационния обект, така че той да не може да бъде прочетен от трети лица. Тази наука приема, че информацията е публична и може да бъде достъпена от всеки, но не може да бъде разбрана. Стеганографията от своя страна има за цел да скрие информационния обект в друг такъв, по такъв начин, че никой друг да не разбере, че в обекта носител има скрита информация. Двете науки са взаимно допълващи се и често се среща комбинация от двете - криптирана информация да е скрита, чрез стеганография в друг информационен обект.

Информационните обекти могат да бъдат различни – изображение, звукозапис, видео запис и др.

Стеганография се използва широко за укриване на информация чрез цифрови файлове. Съобщението може да бъде скрито например в документ, файл, изображение, програма или протокол. Медийните файлове са идеални за тази цел поради големия си размер.

1.2.Исторически примери за използването на стеганография

В историята са известни много и различни примери за използването на стеганография за добри и за лоши цели.

Писането на лист хартия със специално „невидимо“ мастило е било използвано доста често. Този вид техника е бил използван през Втората Световна Война от френската съпротива за комуникация между щабовете си.

Друг пример е за скриването на информация е написан морзов код, чрез завързването на възли върху влакно от прежда. После влакното се прикачало към дреха и така се пренасяла информация.

В наши дни също се използва стеганография. Има сведения, че Ал-кайда (Афганистанска терористична организация) са си комуникирали чрез скрити съобщения в картинки, който после си предавали по електронната поща. Така е бил организиран атентата на 11 септември 2001 година над Световния Търговски Център.

Има и примери от нашето ежедневие. Някой цветни принтери печатат малки жълти точки върху листа хартия, които трудно се забелязват. В тях е скрит серийния номер на принтера.

В *Amazon* използват стеганография за да скрият номера на потребителя в книгата, която си е закупил. На база на интервалите, които са поставени между думите, може да се разбере кой потребител си е закупил електронната книга и да се избегне електронно пиратство.

2. Съществуващи приложения

2.1.VSL – Virtual Steganography Laboratory



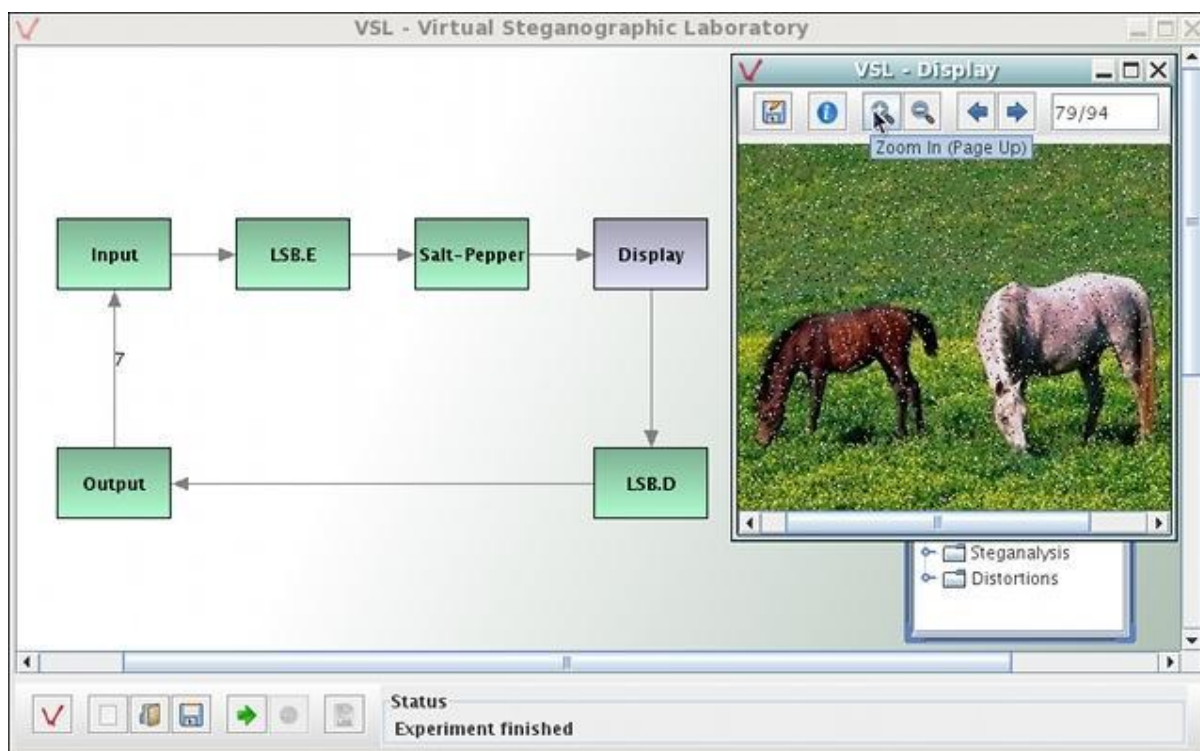
2.1.1. Кратко описание

Това е софтуер, който дава възможност на потребителя да скрие информация в изображения, като използва различни алгоритми. Също има възможността да се направи анализ на вече обработени изображения и да се генерират графики на цветовите разпределения. Предоставя блокова диаграма на потока от логически операции. Така потребителят лесно може да проследи какво се случва с изображението от началото до края на процеса на криене на информация

2.1.2. Поддържани файлови формати

- BMP
- PNG
- JPG
- TIFF

2.1.3. Изображения



фигура 1

2.1.4. Добри страни

- Приложението е написано на езикът *Java*. Така то може да бъде използвано на различни операционни системи - *Windows, Linux, Mac OS X*.
- Разпространява се под *GPLv3* лиценз, което го прави безплатен за употреба.

2.1.5. Лоши страни

- Тъй като е приложение с отворен код, не може да се разчита на постоянна поддръжка, което го прави малко ненадежден
- Изисква се потребителя да има инсталирана *Java* виртуална машина (*JRE - Java Runtime Environment*).

2.2. Open Stego

2.2.1. Кратко описание



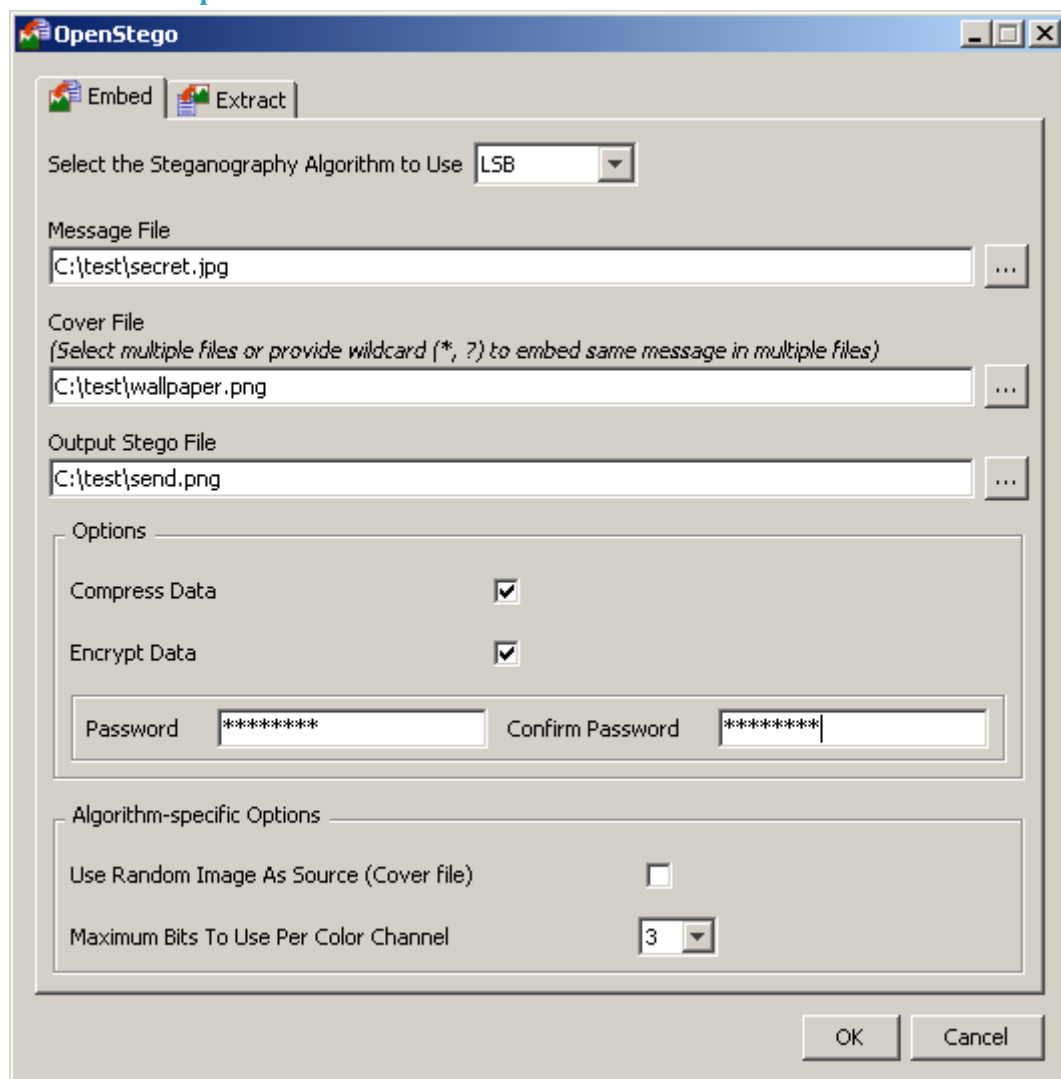
OpenStego е приложение с отворен код, което позволява

скриването и извличането на информация в и от изображение. Позволява допълнително криптиране и компресиране на информацията. Може да бъде сложена парола за допълнителна сигурност.

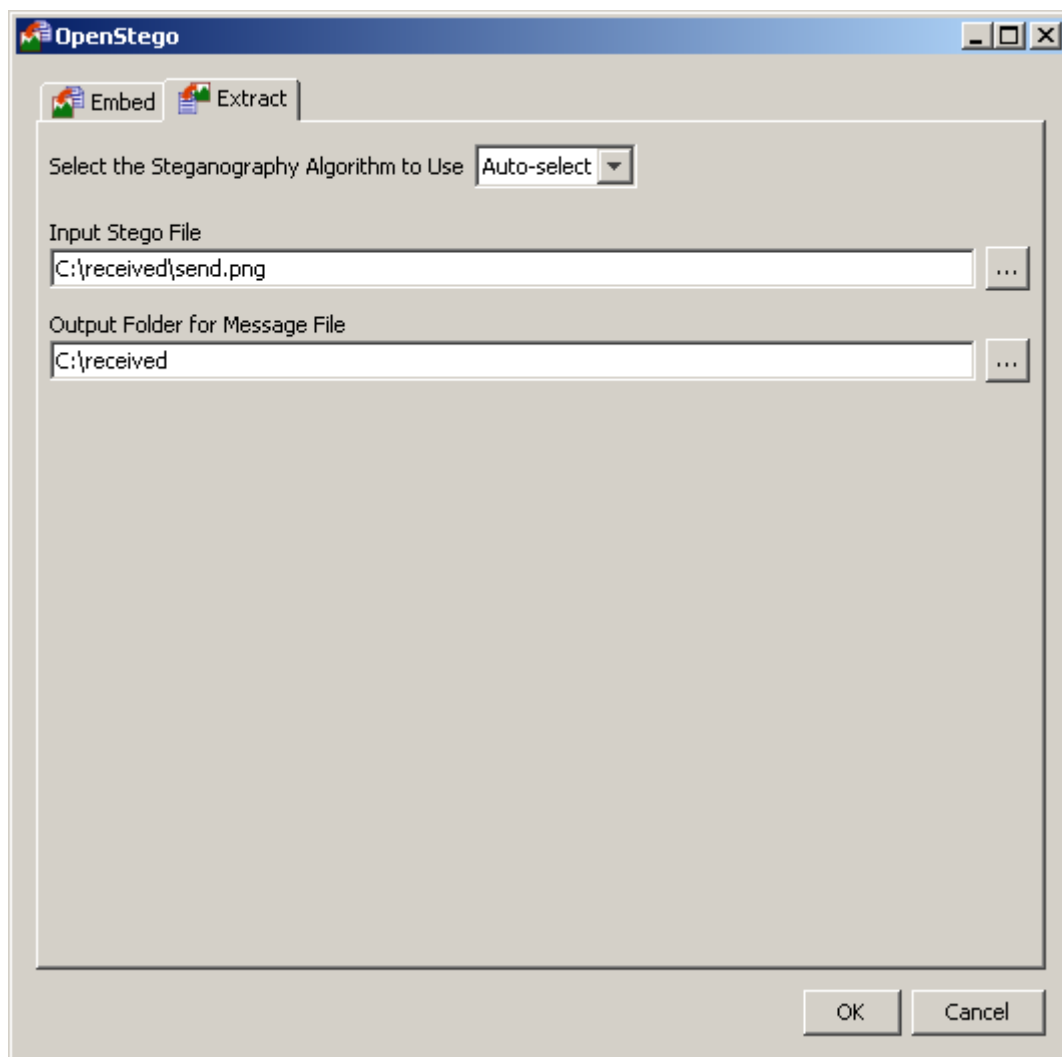
2.2.2. Поддържани файлови формати

- BMP
- PNG

2.2.3. Изображения



фигура 2



фигура 3

2.2.4. Добри страни

- Приложението е написано на езикът *Java*. Така то може да бъде използвано на различни операционни системи - *Windows, Linux, Mac OS X*.
- Разпространява се под *GPLv3* лиценз, което го прави безплатен за употреба.

2.2.5. Лоши страни

- Тъй като е приложение с отворен код, не може да се разчита на постоянна поддръжка, което го прави малко не надежден. Последната версия е била пусната в публичното пространство през 2009 година.
- Изисква се потребителя да има инсталирана *Java* виртуална машина (*JRE - Java Runtime Environment*).

2.3.Stream Steganography

2.3.1. Кратко описание

Stream steganography е библиотека, написана на php. Класът *StreamSteganography* може да бъде използван за скриване и извличане на информация от PNG изображения



2.3.2. Поддържани файлови формати

- PNG

2.3.3. Добри страни

- Има възможност за имплементация на интернет приложение, което да е достъпно от различни места. Това носи платформена независимост и улеснение за клиента (не е нужна инсталация)
- Съществуват PHP интерпретатори за всички популярни операционни системи, което носи допълнителна независимост от към сървърна страна.

2.3.4. Лоши страни

- Библиотеката е извън поддръжка и никой не пише подобрения за нея. Ако се появи евентуален проблем или дефект в библиотеката, няма кой да го оправи.
- Работи само с един файлов формат.

2.4.Steg Hide

2.4.1. Кратко описание

Това е *CLI (Command Line Interface)* приложение. Дава възможността на потребителя да скрива и извлича информация от изображения и аудио файлове. Лесно се използва - с подаването на няколко аргумента към извиканата функция, може да се компресира или конвертира съобщението, което искаме да скрием.

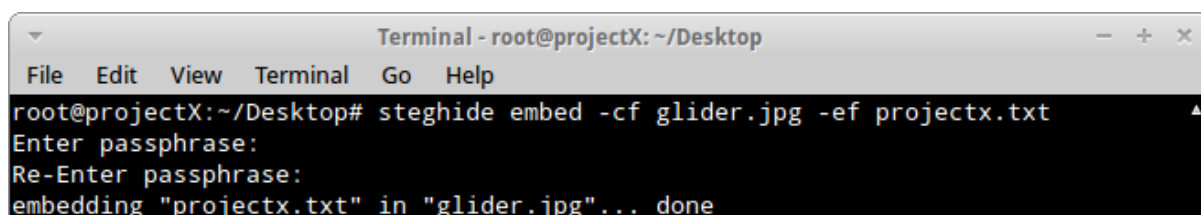


Съществуват и много графични интерфейси, които просто извикват системно *StegHide* с правилните аргументи. Такива са например *SteGUI* (графичен интерфейс за *linux* операционна система) или *StegPHP* (уеб интерфейс написан на php).

2.4.2. Поддържани файлови формати

- JPEG
- BMP
- WAV
- AU files

2.4.3. Изображения



```
Terminal - root@projectX: ~/Desktop
File Edit View Terminal Go Help
root@projectX:~/Desktop# steghide embed -cf glider.jpg -ef projectx.txt
Enter passphrase:
Re-Enter passphrase:
embedding "projectx.txt" in "glider.jpg"... done
```

фигура 4

2.4.4. Външни зависимости

- Zlib
- Libjpeg
- Libmcrypt
- Libmhash

2.4.5. Добри страни

- Лека и лесна за конфигурация програма.
- Може да се използва чрез системно извикване „exec“ в други програми.

2.4.6. Лоши страни

- Работи само под *Linux*
- Не всеки потребител е свикнал да работи с приложения с конзолен интерфейс (*CLI*)

3. Описание на използваните среди и технологии



3.1.Операционна система

Избрах приложението да бъде разработвано и използвано под *UNIX-like* операционна система или по друг начин казано - Linux дистрибуция. Причините бяха няколко:

- Linux операционната система е на първо място софтуер с отворен код и съвсем свободна за употреба. Не е обвързана със закупуване или допълнителни заплащания по поддръжка.
- Стабилен софтуер, идеален за среда на живот на приложение, като моето - трябва да работи постоянно, без да се налага да бъде спирано или рестартирано. Да бъде достъпно по всяко време.
- Безплатна и постоянна поддръжка - много лесно е да отвориш хранилището за пакети и да изтеглиш последна версия на софтуера, който ти е нужен;
- Сигурност - Linux операционните системи са добре защитени и трудно пробиваемы за хакерски атаки;

Дистрибуцията, която аз избрах е *Fedora* версия 17. Тя е разработвана и поддържана от *Red Hat* - гигант в сферата на отворения софтуер. Системата е стабилна, а от *Red Hat* ежедневно доставят подобрения и поддръжка под формата на пакети, които лесно могат да се свалят от публичните хранилища.

3.2.Език за програмиране

Езикът за програмиране, на който е написан стеганографския софтуер е *Java*. Избрах поради няколко причини :



- Мултиплатформеност

За да работи дадено приложение, написано на *Java*, върху определена платформа е нужно единствено да има инсталирана *JVM (Java Virtual Machine)* на нея. Това означава, че то може да бъде пуснато успешно както под *Windows*, така и под *Linux*, *Mac OS X*. Възможна е реализацията дори и върху мобилни платформи като *Android*, тъй като *Android* е писан на *Java*.

- Отворен код

Езикът *Java* бе създаден преди почти 20 години от гигантът *Sun Microsystems*. За щастие той бе разпространен с отворен код, от което последваха различни имплементации на *Java*, които спазват интерфейсът, дефиниран от *Sun* (вече *Oracle*, тъй като *Sun* фалираха и *Oracle* ги купи).

Резултатът е безброй имплементации на Java - всички развиващи се успешно и създадени с различни цели. Пример за такива имплементации са :

- *Oracle Java* - най-разпространената имплементация на Java. Разработена от наследника на *Sun - Oracle*.
- *IBM Java* - имплементация, създадена да работи оптимизирано върху многопроцесорни сървърни машини, разработени от *IBM*.
- *Open Java* - имплементирана от така нареченото *open community* - хора по целия свят, който се занимават със създаване, поддръжка, разпространение и популяризиране на софтуер с отворен код.

➤ Поддръжка

Java е много добре поддържан език. Постоянно се работи върху подобрието на езика и поправянето на съществуващи дефекти. Също съществуват безброй начини да получиш помощ при възникнал проблем - пощенски списъци, форуми, *IRC* канали и други.

➤ Автоматично почистване на “боклука” - *Garbage Collection*

Java използва механизъм, наречен *Garbage collection* или автоматично почистване на “боклука”. Това е механизъм за динамично заделяне, разпределяне и менажиране на виртуалната памет (*heap*). Чрез него се унищожават, неизползваните от приложението, обекти и се освобождава виртуална памет. По този начин разработчика на приложението не се налага ръчно да заделя и освобождава памет за обектите си - това става напълно автоматизирано от *Java* виртуалната машина.

Имплементацията, която аз избрах да ползвам е *Open Java*. Реших да използвам нея, защото е най-добре интегрирана за работа с *UNIX-like* операционни системи. Поддръжката ѝ е постоянна като често има пакети с подобрения и оправени дефекти.

Версията, която използвам е *Java 7* - там са направени много подобрения от страна на натоварване и скорост. Опростени са някои правила в езика. Подобрен е механизма за събиране на “боклук” (*Garbage collection*).

3.3.Интернет сървър



Web сървърът, който е ползваме е *Tomcat*. Той е разработен от *Apache* - общност от разработчици на софтуер, който разпространяват свободен софтуер с отворен код. *Apache* са популярни с поддръжката на различни технологии, приложения и библиотеки, които се ползват във всеобщи сфери :

- *Apache HTTP Server* - *HTTP* сървър, работещ под *Windows* и *Linux*;
- *Apache Wink* - Библиотека за лесна разработка на *RESTfull* сървиси. Написана е на *Java*;

- *Apache Ant* - Java базиран скриптов език, служещ за манипулация на файлове и изпълняване на конзолни команди с цел конвертиране на приложение от код към изпълними файлове;
- *Apache Derby* - Доста популярна база от данни;
- *Apache CouchDB* - База от данни;
- *Apache AXIS* - Платформа за разработка на *XML web* сървиси;
- *Open Office* - Отворена имплементация еквивалент на *Microsoft Office* пакета. Поддържа познатите до момента файлови формати - *.doc, *.docx, *.xls, *.ppt и др.;

Apache Tomcat е Java базиран *web application* сървър, който е лек и с отворен код. Имплементира *Java Servlet* и *Java Server Pages* спецификациите, обявени от *Oracle* корпорацията. Той се състои от три основни части:

- *Catalina* - *servlet* контейнер. Менажира *Java Servlet* обектите. Той отговаря за създаването на инстанция на даден *servlet*, ако е нужно и коректното му унищожаване;
- *Jasper* - *JSP* контейнер. Преобразува *JSP* страниците към *Servlet* обекти, които се подават на *servlet* контейнера. Страниците се обработват, като специфичните за *JSP XML* тагове се трансформират към *Java* код;
- *Coyote* - имплементация на *HTTP* конектор. Отваря, менажира и затваря връзките към сървъра. Поддържа *HTTP* версия 1.1. Слуша на конфигуриран *TCP* порт (по подразбиране 8080), приема заявките към сървъра, разпределя ги към съответния ресурс и връща отговор към клиента;

Apache Tomcat е лесен за конфигурация и употреба. Конфигурацията се осъществява чрез *xml* файл наречен *conf.xml*. В него се описват всички сървърни променливи, портове, виртуални хостове, права за достъп, връзки към бази от данни, начини на аутентикация и др. Много лесно е да се инсталира и интернет приложение - просто се копира *java* интернет архива в папката *webapps* и се рестартира сървъра.

Версията, която ползвам е *Apache Tomcat 7.0*. Там са имплементирани последните версии на *Servlet* и *Java Server Pages* спецификациите, съответно *Servlet 3.0* и *JSP 2.2*. Работи с последна версия на *Java* виртуалната машина - *Java 7*.

3.4. Технологии, използвани при клиента

За реализация на клиентската част на приложението съм използвал браузър зависими езици - *HTML*, *CSS*, *JavaScript*.



Версиите на *HTML* и *CSS* са съответно 4.0 и 2.0. Това не са последните имплементации на *W3* стандарта, но са най-добре поддържаните от всички интернет браузъри. Ако бях използвал последните стандарти - *HTML 5* и *CSS 3*, приложението нямаше да работи коректно във всички

браузъри, поради липсата на имплементация. За момента те са добре поддържани под *Mozilla Firefox* и *Opera*, но за съжаление другите интернет клиенти, като *Microsoft Internet Explorer*, *Safari* и други по-неизвестни такива не могат да се похвалят с тази си функционалност.



Използвал съм *jQuery* - много популярна *javascript* библиотека, която унифицира работата с езика, като предоставя интерфейс, който е браузър независим. Разработката на библиотеката стартира в далечната 2006 година и към момента е най-популярната и най-използваната в света. Към нея съм добавил и два плъгина:

- *jQueryPostForm* - използва се за *AJAX submit* към сървъра. Така се избягва презареждане на цялата страница, а вместо това се променя само *form* елемента, ускорява се производителността, чрез олекотяване на *HTTP* заявката;
- *jQueryUI* - използва се за придаване на по-красив вид и добавяне на нови функционалности към *HTML* контролите. С негова помощ са създадени *input*, *radio*, *select* и *tab* контролите;

3.5. Външни зависимости

3.5.1. На сървъра

Приложението няма външни зависимости. За да работи са нужни операционна система с инсталиран и конфигуриран на нея *Apache Tomcat web* сървър, и инсталирано на нея приложението

3.5.2. При клиента

При клиента единствената нужна конфигурация е да бъде разрешено използването на *javascript* в интернет браузъра. Тази настройка по подразбиране е пусната във повечето популярно браузъри, а и е нужна за работата на много други известни сайтове.

4. Избор на алгоритъм

За целите на приложението е избран алгоритъма *LSB (Least significant bit)*, който ще бъде разгледан по-долу. Алгоритъмът бе избран защото е един от основните стеганографски алгоритми и ясно изразява идеята на стеганографията (както ще се види малко по-късно) за скриване на информационен обект в друг такъв.

4.1.Описание на алгоритъма

Както се разбира от името му, *LSB* алгоритъма използва най-малко значимия бит на всеки байт информация на обекта носител, за да скрие един бит от информацията на обекта, който трябва да бъде скрит. Евентуалната малка промяна в цвета на даден пиксел е незабележима за човешкото око, но е достатъчна за да носи част от скритата информация.

4.1.1. Побитови операции

LSB алгоритъма използва двоични операции за да обработва битовите информация. Използват се следните логически операции:

➤ OR

❖ Таблица на стойностите

Ляв операнд	Десен операнд	Резултат
0	0	0
0	1	1
1	0	1
1	1	1

таблица 1 – Таблица на стойностите на OR логическа операция

❖ Пример

	Двоично представяне	Десетично представяне
Ляв операнд	01010111	87
Десен операнд	01100101	101

Резултат от OR операцията	01110111	119
---------------------------	----------	-----

таблица 2 – Примери за OR логическа операция

➤ AND

❖ Таблица на стойностите

Ляв операнд	Десен операнд	Резултат
0	0	0
0	1	0
1	0	0
1	1	1

таблица 3 – Таблица на стойностите на AND логическа операция

❖ Пример

	Двоично представяне	Десетично представяне
Ляв операнд	01010111	87
Десен операнд	01100101	101
Резултат от AND операцията	01000101	69

таблица 4 – Примери за AND логическа операция

➤ LEFT SHIFT

Преместването наляво е операция, при която всички битове се изместват с една позиция наляво, като най-дясната позиция се запълва с 0.

❖ Пример

	Двоично представяне	Десетично представяне	Брой премествания	Резултат (BIN)	Резултат (DEC)
Пример №1	01010111	87	1	10101110	174
Пример №2	01010111	87	2	01011100	92

таблица 5 – Примери за LEFT SHIFT логическа операция

➤ RIGHT SHIFT

Операцията е обратна на преместване наляво. Всички битове се изместват с една позиция надясно, а на най-лявата позиция се добавя 0.

❖ Пример

	Двоично представяне	Десетично представяне	Брой премествания	Резултат (BIN)	Резултат (DEC)
Пример №1	01010111	87	1	00101011	43
Пример №2	01010111	87	2	00010101	21

таблица 6 – Примери за RIGHT SHIFT логическа операция

4.1.2. Последователност на алгоритъма

Нека наричаме информацията, която искаме да скрием “съобщение”, а обекта, в който искаме да я скрием - “носител”. LSB алгоритъма използва описаните по-горе операции за да запише информация в обекта носител. Двата информационни обекта се представят в двоичен вид - масив от битове. Всеки бит от съобщението се записва в най-малко значимия бит на последователните байтове на носителя.

Алгоритъма за записване на дадено двоично съобщение в двоичен носител следва долния псевдокод:

```

За всеки байт от съобщението
{
    var byte = текущия байт;
    За всеки бит от текущия байт
    {
        var index = Индекса на текущия бит;
        var tmp = (byte >> index) & 1;
        var dest =байта, в който искаме да скрием текущия бит;
    }
}

```

```

    dest = dest & 0xFE | tmp;
}
}

```

Ето пример как работи алгоритъма по стъпки. Така най-ясно се вижда какво точно се случва с битовете и байтовете на съобщението и носителя:

Нека имаме дадени следните стойности за съобщение и носител:

Съобщение	11010001
Носител	100101100101011011010110
	110101101100011011011110
	1001001101010110

таблица 7 – Примерни входни данни (съобщение и носител)

1. Взимаме последователно всеки бит от текущия байт на съобщението

Ако искаме да вземем i -тия бит, използваме следните операции:

- Операция SHIFT RIGHT i на брой пъти върху байта;
- Резултата го умножаваме по двоична единица;

```

var index = Индекса на текущия бит;
var tmp = (byte >> index) & 1;

```

	Съобщение	Индекс	Резултат	1 (BIN)	Резултат
Пример №1	11010001	3	00011010	00000001	00000000
Пример №2	11010001	6	00000011	00000001	00000001

таблица 8 – Пример за взимане на текущия бит от съобщението

2. За всеки бит от съобщението взимаме последователни групи от по 8 бита (1 байт)

Носителят е поредица от битове. Всяка група от 8 последователни бита образува един байт. Всеки бит от съобщението се скрива в 8-мия бит на всеки байт. За целта разбиваме носителя на групички от по 8 бита:

Индекс	Стойност
0	10010110
1	01010110
2	11010110
3	11010110
4	11000110
5	11011110
6	10010011
7	01010110

таблица 9 – Пример за разбиване на носителя на байтове от по 8 бита

3. Записваме текущия бит на съобщението в “най-малко значимия” бит от текущия байт на носителя
 - Умножаваме текущия байт с 0xFE (11111110). По този начин нулираме “най-малко значимия” бит от байта
 - Изпълняваме побитово “или” върху получената стойност със стойността на текущия бит. По този начин битът се записва в последния бит от байта

var dest = байта, в който искаме да скрием текущия бит;
dest = dest & 0xFE | tmp;

Индекс	Байт	0xFE	Бит	Резултат
0	10010110	11111110	1	10010111
1	01010110	11111110	0	01010110
2	11010110	11111110	0	11010110
3	11010110	11111110	0	11010110
4	11000110	11111110	1	11000111

5	11011110	11111110	0	11011110
6	10010011	11111110	1	10010011
7	01010110	11111110	1	01010111

таблица 10 – Пример за записване на текущия бит на съобщението в най-малко значимия бит от текущия байт на носител

4. Резултат

Носител	
Преди	1001011001010110110101101101011011000110110111101001001101010110
След	1001011101010110110101101101011011000111110111101001001101010111

таблица 11 – Примерни резултати

Всъщност данните, които се записват в носител са две - дължината на скритото съобщение и самото съобщение. По този начин при декодиране знаем колко точно символа да дешифрираме от носител. Това обаче налага ограничение на дължината на съобщението. За да прочетем дължината, трябва да знаем къде е записана и в колко байта.

За имплементацията в това приложение съм взел решение дължината на съобщението да се записва в 32 байта. Това означава, че съобщението има ограничена дължина до 9999 символа. Байтовете, в които се записва дължината се променят динамично спрямо парола, която може да бъде въведена от потребителя. Това е обяснено по-подробно в следващите точки.

Друго ограничение, което е наложено с цел смалване на съобщението е, че софтуерът работи само със символи от ASCII таблицата. Тъй като тези символи са 8 битови и отнема много по-малко място да се скрият от колкото символи, кодирани в UNICODE например.

4.1.3. Подобрения

За по-добро скриване на съобщението съм добвил възможност за динамично определяне на първия байт, в който ще се записва съобщението. Това става чрез парола, която потребителя има право да въведе. Върху паролата се изпълнява хеш функция, която връща цяло неотрицателно число. Това число показва индекса на байта от който да започне скриването на съобщението. При декодиране на потребителя отново му се дава възможност да въведе парола. Ако паролата не съвпада с тази, която е въведена при скриването на съобщението, то няма да може да бъде намерено, тъй като стартовия байт ще е различен.

Разбира се винаги има възможност за колизии при изчисляването на хеш стойността (при изпълнение на една и същата хеш функция върху различни входни параметри да се получи един същ хеш резултат). Това не може да бъде избегнато, но може да бъде намалена вероятността за получаване на такива. Ако се използва по-сложна хеш функция, вероятността за получаване на колизии е много по-малка.

4.1.4. Обратната посока

След като сме намерили хеша на паролата и знаем от кой поред байт да започнем да извличаме информация, трябва да намерим дължината на съобщението. Нека наричаме стойността на хеша “стартов байт”.

Дължината на съобщението е скрива в първите 32 байта разположени след стартовия байт. За да намерим дължината трябва да вземем най-малко значимия бит от всеки байт в този интервал и да конструираме нови байтове, който представляват дължината на съобщението.

За всеки байт от интервала трябва да изпълним следните операции:

- Създаваме си променлива *length*, в която да записваме битовете, който сме извлекли от байтовете.
- Умножаваме всеки байт с двоична единица, за да получим стойността на най-малко значимия бит
- Изпълняваме “преместване наляво” върху *length* един път, за да имаме свободен бит, в който да запишем извлечената стойност
- Изпълняваме логическа операция “или” върху *length* и извлечената стойност, за да запишем новия бит в *length*

❖ Пример

Нека имаме дадени следните два байта и променлива *length*:

Променлива	Стойност
<i>length</i>	00000000
Byte №1	11010101
Byte №2	11010110

таблица 12 – Примерни входни данни

Умножаваме байтовете с двоична единица, за да получим най-малко значимия бит:

	Стойност	Единица (BIN)	Резултат
Byte №1	11010101	00000001	1
Byte №2	11010110	00000001	0

таблица 13 – Пример за взимане на най-малко значимия бит от байта на носителя

Изпълняваме операцията LEFT SHIFT върху променливата *length* и след това я събираме двоично с резултата от горната таблица. Това се прави последователно за всеки резултат.

Номер на байта	Length	Стойност	Резултат
1	00000000	1	00000001
2	00000001	0	00000010

таблица 14 – Пример за добавяне на извлечения бит от носителя, към съобщението

По аналогичен начин се изчисляват и останалите битове от стойността на променливата *length*. След като знаем колко байта е дължината на скритото съобщение, можем по същия алгоритъм да продължим да извличаме и самото съобщение. За него няма да давам примери, тъй като алгоритъмът е аналогичен с показания по-горе.

5. Реализация на стеганографския софтуер

5.1.Избор на решение

Приложението, което съм имплементирал представлява интернет приложение, което се достъпва през браузър. То представлява интернет страница, на която могат да се качват и смъкват изображения, в/от които да може да се добави/извлече съобщение.

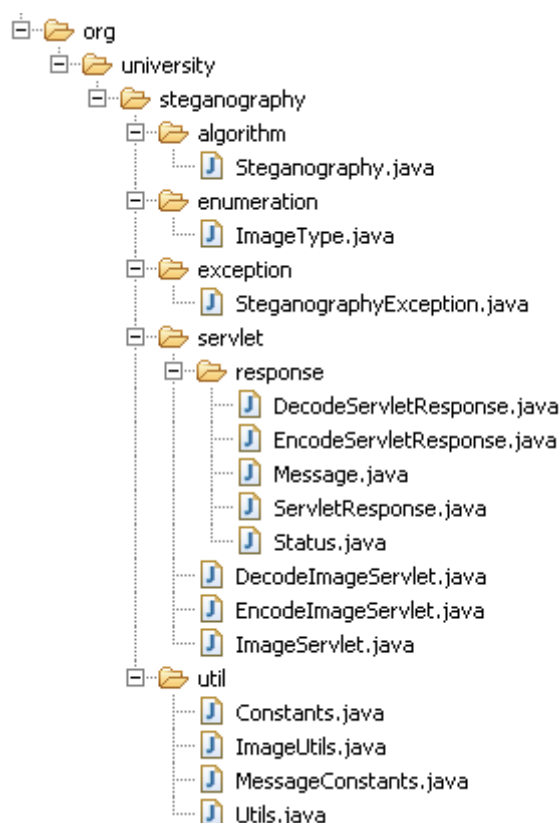
Приложението работи с AJAX и страницата не се презарежда при различните операции. Това я прави удобна за вграждане в други сайтове с образователна цел.

Друг плюс е, че е лесно за употреба и е мултиплатформено. Ако човек иска да го използва, просто трябва да напише интернет адреса в браузъра си, не е нужно да тегли и инсталира различни допълнителни приложения. Също така може да бъде използвано под различни операционни системи, дори на различни устройства (настолни, мобилни, дори вградени)

5.2.Класова йерархия

Класовете в приложението са разделени логически в отделни *Java* пакети. Пакетите са следните:

- **org.university.steganography.algorithm** - В него са разположени класовете, които се имплементират алгоритми за стеганография. За момента е имплементиран само LSB алгоритъм, но има възможност за бъдещо развитие, като се добавят нови имплементации;
- **org.university.steganography.enumeration** - Тук са разположени всички изброени типове, нужни за проекта;
- **org.university.steganography.exception** - Тук се намират различните видове Exception класове, които се използват в приложението;
- **org.university.steganography.servlet** - Тук се намират имплементациите на различните *servlet* класове. Те приемат заявките, които пристигат към сървъра и ги обработват по подходящ начин;
- **org.university.steganography.servlet.response** - Тук се намират класове, които представляват различните видове отговор от сървъра към клиента. Те са нужни за да има ясен и структуриран отговор, който да може да бъде обработен при клиента от AJAX обработчика;
- **org.university.steganography.util** - Тук се намират класовете, които могат да бъдат наречени "помощни". Те се използват във останалите класове и извършват операции, които се повтарят на много места и помагат кода да изглежда по-четим;



фигура 5 – Йерархия на класовете, разпределени по директории

5.3.Описание на основните класове

5.3.1. Steganography

Изпълнява операциите, свързани със скриването на информация и извличането на такава в/от подадено изображение. Има няколко метода, които имплементират логиката за *LSB* алгоритъма, описана по-горе.

➤ **private** BufferedImage encodeText

Това е метод, който скрива съобщение, представено в формат на масив от байтове в изображение. Като входни параметри се подават изображението (носителя на информация), съобщението (във формата на масив от байтове) и стартовия байт, който прадварително е изчислен на база парола, подадена от потребителя.

Формата, в който е изображението е *java.awt.BufferedImage*. Това е клас от *Java* интерфейса, който дава възможност за обработка на различни изображения. Методите, които използвам от този клас са *getRGB* и *setRGB*, които съответно взимат стойността на даден елемент от цветовата матрица на изображението или я променят на зададената.

Първоначално се изчислява размера на изображението (височина и ширина). Тези данни са нужни за да се обходи коректно матрицата със цветови стойности.

```
final int height = image.getHeight();
final int width = image.getWidth();
```

Обхождането става по редове.

```
int i = offset / width;
int j = offset % width;
if ((width * height) >= (addition.length * 8 + offset))
{
    for (final byte add : addition)
    {
        for (int bit = 7; bit >= 0; --bit)
        {
            ....
        }
    }
}
```

Всеки бит от байта се записва като последен бит в байта на изображението.

```
final int imageValue = image.getRGB(i, j);
int b = (add >>> bit) & 1;
final int imageNewValue = ((imageValue & 0xFFFFFFFE) | b);
image.setRGB(i, j, imageNewValue);
```

➤ **private byte[]** decodeText

Това е метод, който извлича скрито съобщение от подаденото му изображение. Като входни параметри приема изображението, което трябва да се декодира и старториен бит, изчислен предварително на база паролата, подадена от потребителя. Като резултат от изпълнението на метода се връща масив от байтове, който представлява скритото съобщение.

Методът се състои от две части : първа - в която извличаме дължината на скритото съобщение и втора - в която извличаме скритото съобщение. Кодът отдолу представлява частта с изчисляване на дължината на скритото съобщение:

```
final int width = image.getWidth();
final int offset = startingOffset +
Constants.HIDDEN_MESSAGE_BIT_LENGTH;
int length = 0;
for (int i = startingOffset; i < offset; ++i)
{
    final int h = i / width;
    final int w = i % width;

    final int imageValue = image.getRGB(h, w);
    length = (length << 1) | (imageValue & 1);
}
```

Първоначално се изчислява интервала от байтове, в който се намират скритите битове на дължината на съобщението. После от всяка стойност се взима последния бит и се добавя към променливата length (добавянето става с побитова операция LEFT SHIFTH).

5.3.2. ImageType

Това е изброен тип, който показва какви са познатите файлови формати.

5.3.3. SteganographyException

Генерална грешка, която се хвърля от приложението, ако нещо некоректно се случи. Има няколко конструктора в зависимост от това дали се подава съобщение за грешка или самата грешка.

5.3.4. ServletResponse

Абстрактен клас, представляващ *HTTP* отговора от сървъра към клиента. Всички *servlet* обекти трябва да връщат отговор от такъв тип. Обектът се сериализира до *JSON (JavaScript Object Nation)* за да бъде лесен за обработка в последствие от клиента. Съдържа едно поле от тип *Status*.

5.3.5. EncodeServletResponse

Клас, разширяващ абстрактния клас *ServletResponse*. Използва се за да върне отговор от сървъра към клиента при извикване на *EncodeImageServlet* сървлета. Няма собствени полета, използва само наследените (status от тип *Status*) от абстрактния клас.

5.3.6. DecodeServletResponse

Клас наследяващ абстрактния клас *ServletResponse*. Представява *HTTP* отговора от сървъра към клиента при извикване на *DecodeImageServlet* сървлета. Съдържа едно допълнително поле *message* от тип *String*, в което се пренася скритото съобщение.

5.3.7. Message

Обектите от този клас се подават на към клиента в отговора от сървъра. Те дават информация за евентуално настъпили грешки при обработката на информацията, предупреждения за нещо, което не е коректно или просто връщат някакво информационно съобщение към клиента.

Класът има две полета от тип *String* :

- *type* – Това е видът на съобщението. Може да бъде информационно, предупредително или грешка;
- *text* – Тук се пази текстът на съобщението;

5.3.8. Status

Клас, който се използва за да върне статус за изпълнената операция към клиента, направил *HTTP* заявката. Така той разбира дали всичко е минало успешно или ако не е - получава адекватни съобщения за настъпилата ситуация. Класът има две полета и един метод:

- *success* – Булева променлива, която индикира дали операцията е протекла успешно или са настъпили грешки по време на изпълнението ѝ;
- *messages* – Списък със съобщение от тип *Message* (описан по-горе), носещи информация от сървъра към клиента;
- *addMessage* – Методът служи за добавяне на съобщения към списъка. Препоръчва се неговото ползване вместо ръчно да се добавят съобщения към списъка, тъй като ако съобщението е тип грешка, методът ще вдигне флага *success*;

5.3.9. EncodeImageServlet

EncodeImageServlet наследява класа *Servlet* от *Java* интерфейса. *Servlet* е обект, който се менажира от контекста на сървъра (*context managed*). Това означава, че интернет сървърът отговаря за създаването, извикването и унищожаването на инстанцията му. *Servlet* обектите се качат за определен интернет адрес и когато сървърът получи заявка за този адрес проверява дали има създадена инстанция на сървлета, който трябва да я обработи. Ако няма – създава нов обект, ако има – заявката се подава на метода *service*. Метода *service* проверява какъв е типа на *HTTP* заявката – *GET*, *POST*, *HEAD* и др. За всяка една от деветте *HTTP* заявки има метод в класа *Servlet*, който го обработва. Например за заявка от тип *GET*, методът е *doGet*. За заявки от тип *POST*, методът е *doPost* и т.н. *EncodeImageServlet* поддържа само *HTTP POST* заявки. За това е предефиниран само метода *doPost*.

В началото на метода се инициализират променливите за статус и се взимат параметрите на *HTTP* заявката подадена към сървъра. След това на база на параметъра „*encode-source*“ се взима решение от къде да се прочете изображението – от интернет адрес или е подадено като параметър към заявката. След като всички променливи са инициализирани, се пресмята какъв е индекса на стартовия байт. Това става на база паролата, която потребителя е подал на заявката. След това се извиква метода *encode* от класа *Steganography* (описан по-горе) и ако всичко мине успешно, новото и старото изображение се запазват в сесията на потребителя. Това се прави с цел те да могат да бъдат показани в последствие на потребителя и той да може да ги сравни или да ги свали локално при него. Ако се е случило нещо непредвидено по време на изпълнение на горните операции, се добавя съобщение за грешка към променливата статус. След това тя се сериализира до *JSON* обект и се подава на отговора към клиента.

5.3.10. DecodeImageServlet

DecodeImageServlet класа отново наследява *Servlet* класа от *Java* интерфейса. Той обработва само *HTTP* заявки от тип *POST*, за това имплементира само метода *doPost*.

Първоначално се инициализират променливите и се взимат параметрите от *HTTP* заявката. След това на база на параметъра „*decode-source*“ се взима решение от къде да се прочете изображението – от интернет адрес или е подадено като параметър към заявката. След това на база на големината на изображението и паролата, която потребителя е подал на заявката, се изчислява индекса на стартовия байт. След това се вика метода *decode* от класа *Steganography*, който ако се изпълни успешно, връща скритото съобщение от изображението, ако не – съобщение за грешка е добавено към статуса. Съобщението заедно със статуса се сериализират до *JSON* обект и се подават на отговора към клиента.

5.3.11. ImageServlet

ImageServlet класа отново наследява *Servlet* класа от *Java* интерфейса. Той обработва само *HTTP* заявки от тип *GET*, за това имплементира само метода *doGet*. Работата на този сървлет е да върне в двоичен вид едно от изображенията –оригиналното или кодираното към клиента. В началото на метода се прочита параметъра „*version*“, който показва кое изображение е поискано от клиента. Изображението се зарежда от сесията на потребителя и се подава на отговора към клиента.

5.3.12. Constants

Съдържа константи, използвани в приложението. Добра практика е да няма така наречените “магически числа”. Всяка константа, е добре да бъде изнесена във клас или интерфейс (какво в случая), за да може да бъде ползвана на повече от едно място и да има говорящо име на променливата. Така се изчиства имплементираната логика и кодът става по-четим.

5.3.13. ImageUtils

Съдържа помощни методи за обработка на изображение. Тези методи се ползват най-вече в класа *Steganography* при скриване или извличане на съобщение в/от картинка

- *hasAlpha* – Приема един параметър от тип *Image*. Проверява дали изображението има алфа пиксели (чрез такива пиксели се реализира прозрачност на изображението).
- *getImageMimeType* – Приема един входен параметър от тип *File*. Проверява *MIME* типът на файла – дали е текстов документ, изображение, изпълним файл и др.
- *getImageType* – Приема един входен параметър от тип *String*. Връща стойност от тип *ImageType* (описан по-горе) като проверява дали подаденият стринг е един от типовете на изображение, дефинирани в изброения тип.

5.3.14. MessageConstants

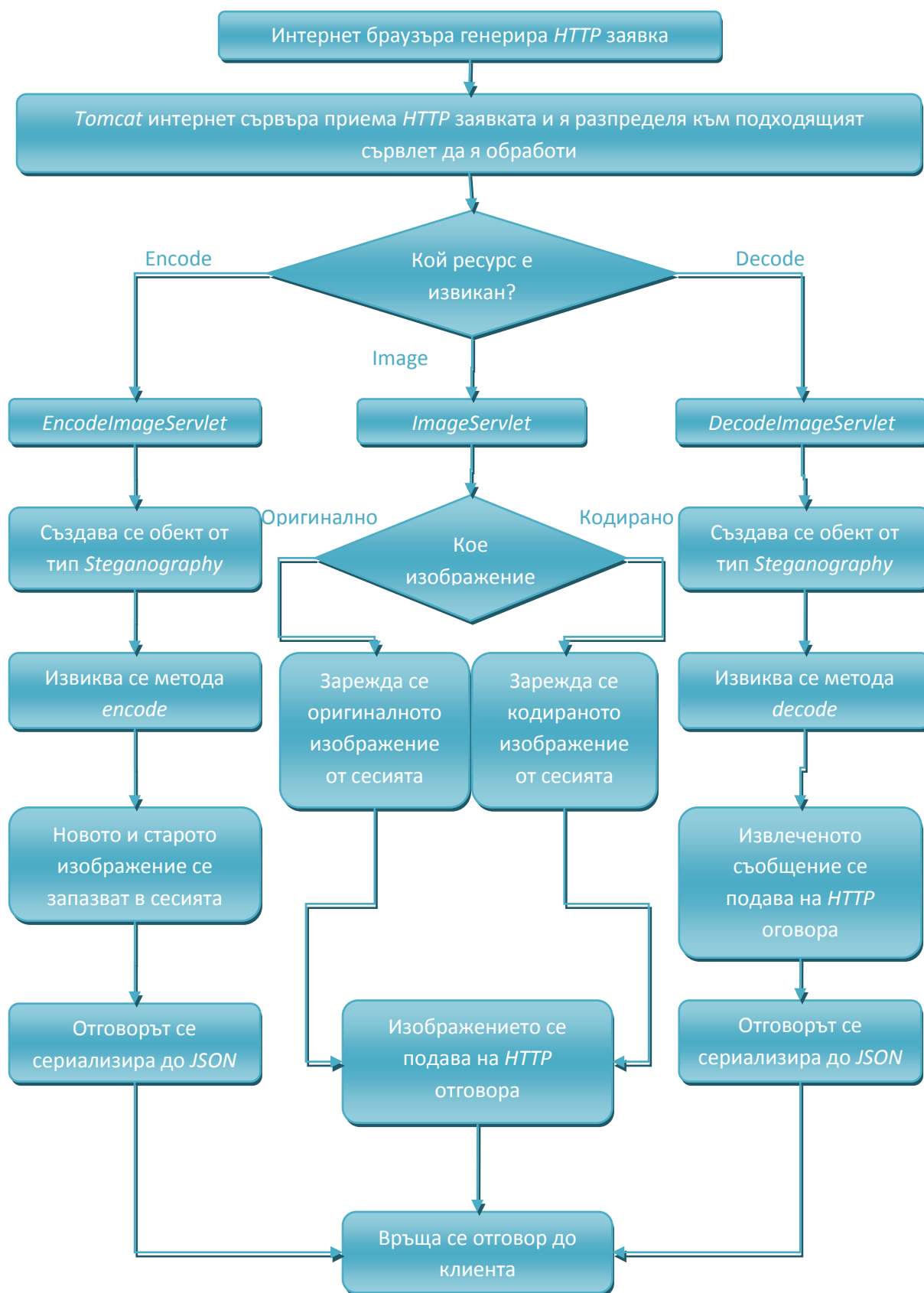
Съдържа съобщения, използвани в приложението

5.3.15. Utils

Съдържа набор от помощни методи, които се ползват на много места в приложението.

- *isEmpty* – Приема единствен входен елемент от тип *String*. Връща булева стойност *true*, ако параметърът е различен от *null* и не е празен стринг.
- *streamToImage* – Приема единствен параметър от тип *InputStream*. Извлича изображение от потока от данни и връща обект от тип *BufferedImage*.
- *getFilename* – Приема единствен параметър от тип *Part* (това е типа на данните, които се подават към сървъра при изпращане на изображение). Връща името на файла.
- *getImageAsBytes* – Приема единствен параметър от тип *BufferedImage*. Връща изображението представени като масив от байтове.
- *compareImages* – Приема два аргумента, и двата от тип *BufferedImage*. Сравнява двете изображения байт по байт и връща булева стойност *true*, ако всички битове на изображенията са еднакви.
- *loadRemoteFile* – Приема единствен параметър от тип *String*. Зарежда изображение от отдалечен ресурс. Ресурсът се представя от входния параметър, като това е неговия интернет адрес. Връща обект от тип *BufferedImage*.
- *calculateStartingOffset* – Приема два параметъра – един от тип *String* и един от тип *Long*. Изчислява индекса на началния байт в изображението за декодиране на база подадените му парола и размер на изображение.

5.4.Описание на работата на приложението



фигура 6 - Блок схема на работата на приложението

6. Тестове

Тестовите, който направих, сравняват оригиналното изображение с това, което е кодирано със съобщение, използвайки приложението. Изображенията са сравнявани на база следните параметри:

- Видими разлики – сравняване на двете изображения преди и след кодирането;
- Размер – размер на изображението в байтове;
- Брой уникални цветове – това е броят на нюансите на цветовете, които са използвани в изображението;
- Има ли изображението прозрачност – дали изображението поддържа алфа блендинг;
- Хистограми – графика на разпределението на червен, зелен и син цвят;

6.1.Тест №1



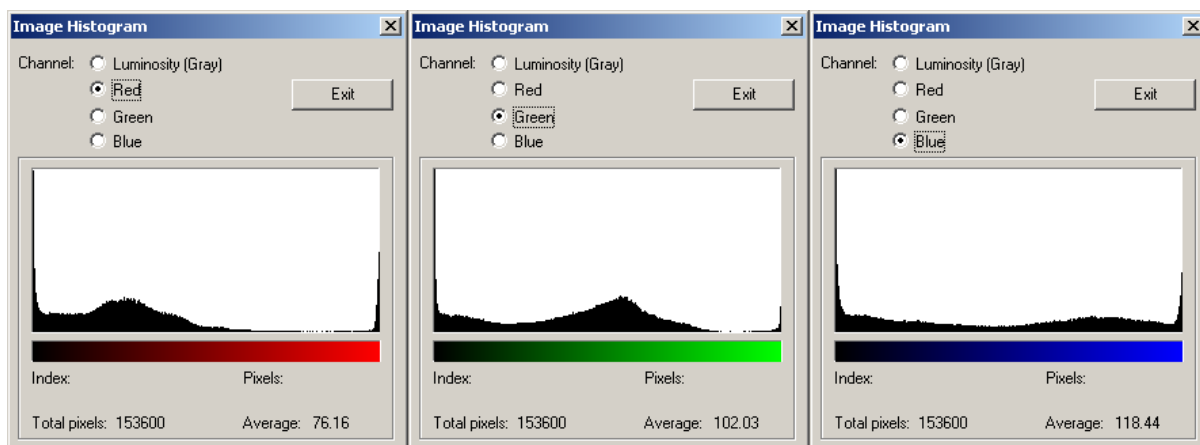
фигура 7 – Оригинално изображение



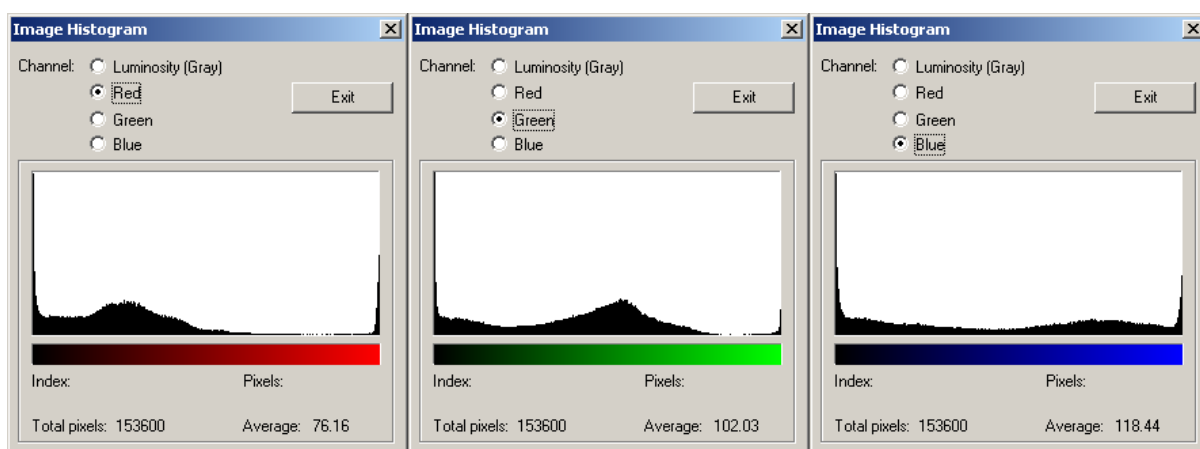
фигура 8 – Кодирано изображение

На фигура 7 е показано изображението на папагал, преди да бъде кодирано със скрито съобщение.

На фигура 8 е показано същото изображение, но с кодирано в него съобщение.



фигура 9 – Хистограма на разпределението на червен, зелен и син цвят в оригиналното изображение



фигура 10 - Хистограма на разпределението на червен, зелен и син цвят в кодираното изображение

На фигура 9 са показани хистограмите за червен, зелен и син цвят на оригиналното изображение, показано на фигура 7.

На фигура 10 са показани хистограмите на червен, зелен и син цвят на кодираното изображение, показано на фигура 8.

	Оригинално изображение	Кодирано изображение
Размер (Байтове)	420 817	420 809
Брой уникални цветове	124 709	124 709
Резолюция (пиксел x пиксел)	320 x 480	320 x 480
Дължина на паролата	-	15
Дължина на съобщението	-	69

Има ли прозрачност?	Не	Не
---------------------	----	----

таблица 15 – Сравнителни характеристики

6.2.Тест №2



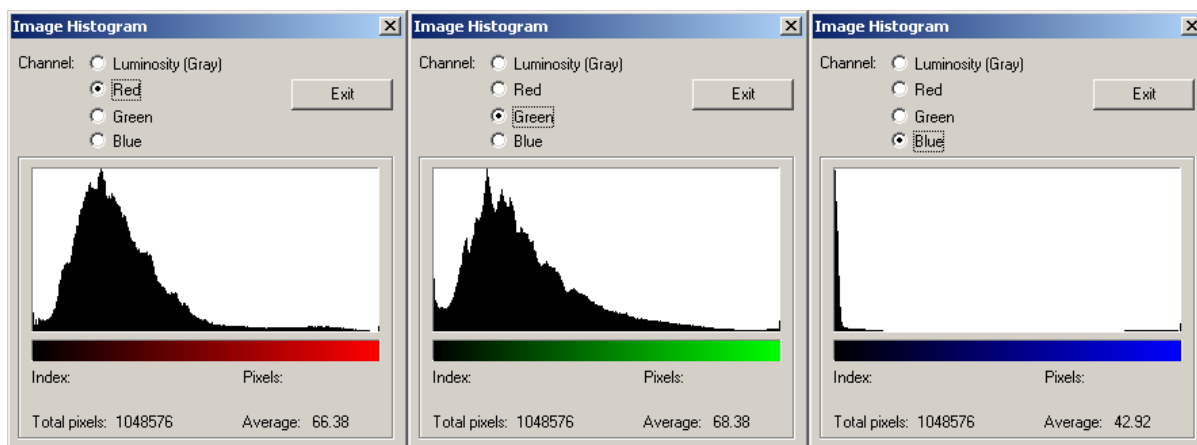
фигура 11 – Оригинално изображение



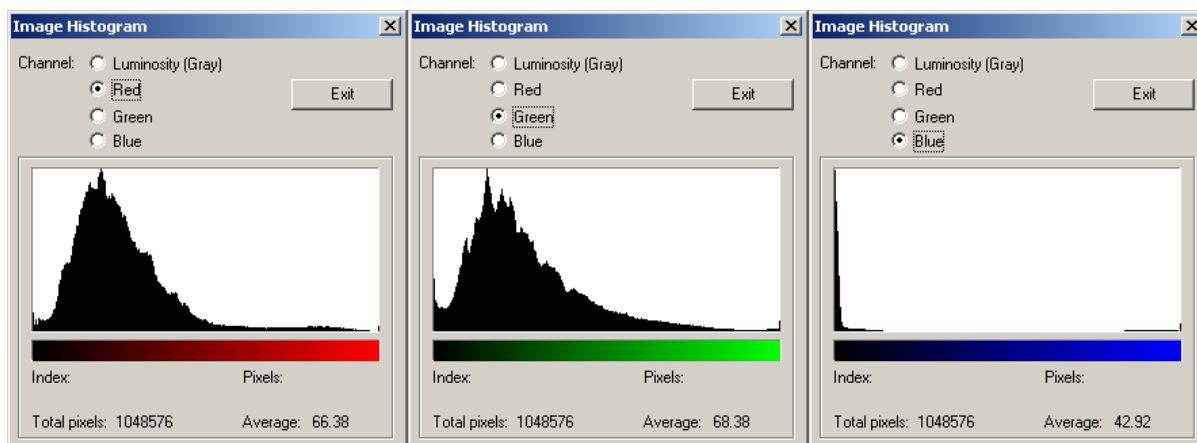
фигура 12 – Кодирано изображение

На фигура 11 е показано изображението на папагал, преди да бъде кодирано със скрито съобщение.

На фигура 12 е показано същото изображение, но с кодирано в него съобщение.



фигура 13 - Хистограма на разпределението на червен, зелен и син цвят в оригиналното изображение



фигура 14 - Хистограма на разпределението на червен, зелен и син цвят в кодираното изображение

На фигура 13 са показани хистограмите за червен, зелен и син цвят на оригиналното изображение, показано на фигура 11.

На фигура 14 са показани хистограмите на червен, зелен и син цвят на кодираното изображение, показано на фигура 12.

	Оригинално изображение	Кодирано изображение
Размер (Байтове)	1 663 294	1 663 714
Брой уникални цветове	203 159	203 273
Резолюция (пиксел x пиксел)	1024 x 1024	1024 x 1024
Дължина на паролата	-	15
Дължина на съобщението	-	69
Има ли прозрачност?	Не	Не

таблица 16 – Сравнителни характеристики

6.3.Тест №3



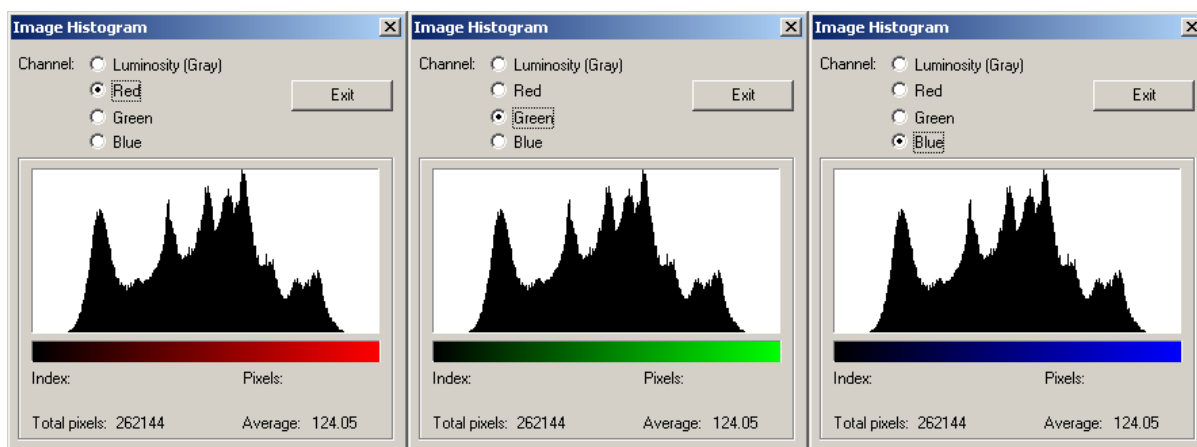
фигура 15 – Оригинално изображение



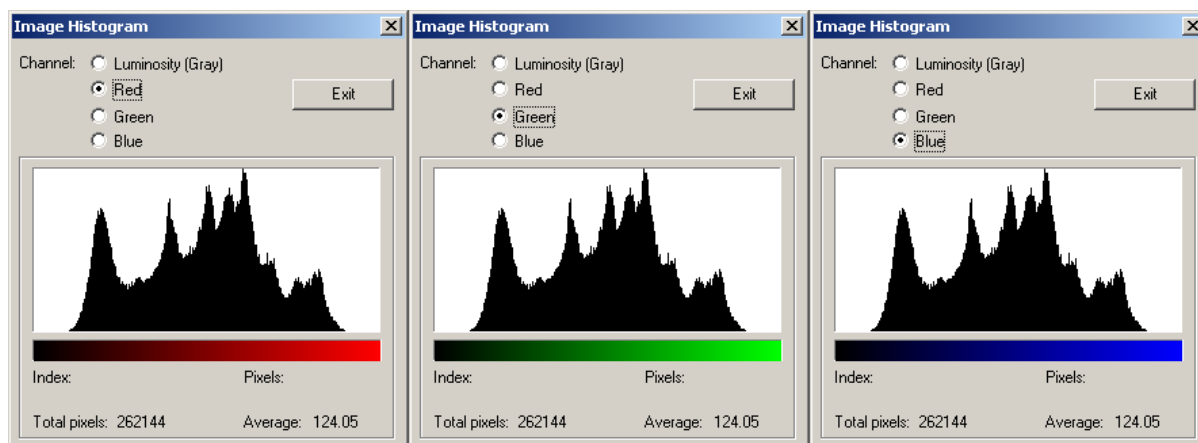
фигура 16 – Кодирано изображение

На фигура 15 е показано изображение на жена, преди да бъде кодирано със скрито съобщение.

На фигура 16 е показано същото изображение, но с кодирано в него съобщение.



фигура 17 - Хистограма на разпределението на червен, зелен и син цвят в оригиналното изображение



фигура 18 - Хистограма на разпределението на червен, зелен и син цвят в кодираното изображение

На фигура 17 са показани хистограмите за червен, зелен и син цвят на оригиналното изображение, показано на фигура 15.

На фигура 18 са показани хистограмите на червен, зелен и син цвят на кодираното изображение, показано на фигура 16.

	Оригинално изображение	Кодирано изображение
Размер (Байтове)	247 766	248 522
Брой уникални цветове	231	322
Резолюция (пиксел x пиксел)	512 x 512	512 x 512
Дължина на паролата	-	12
Дължина на съобщението	-	65
Има ли прозрачност?	Не	Не

таблица 17 – Сравнителни характеристики

6.4.Тест №4



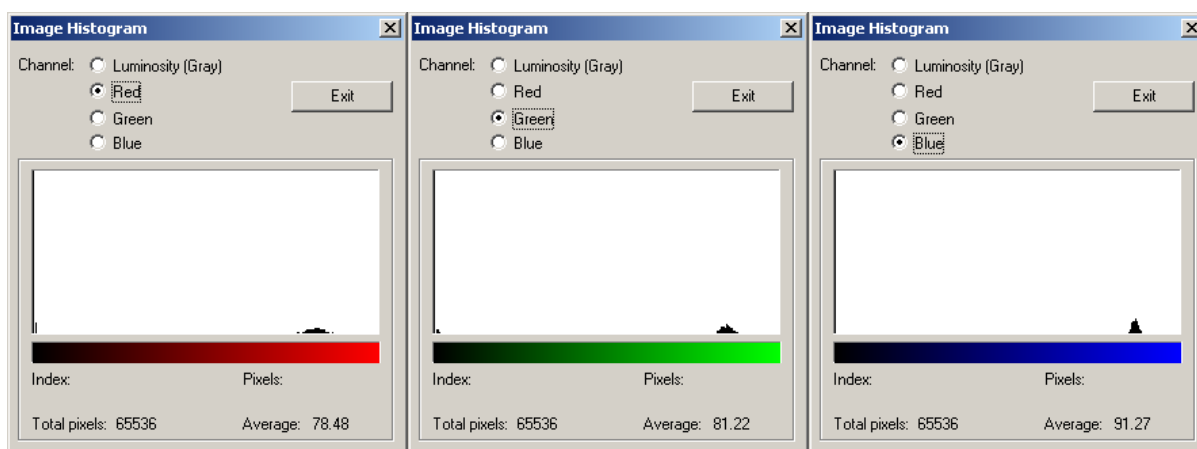
фигура 19 – Оригинално изображение



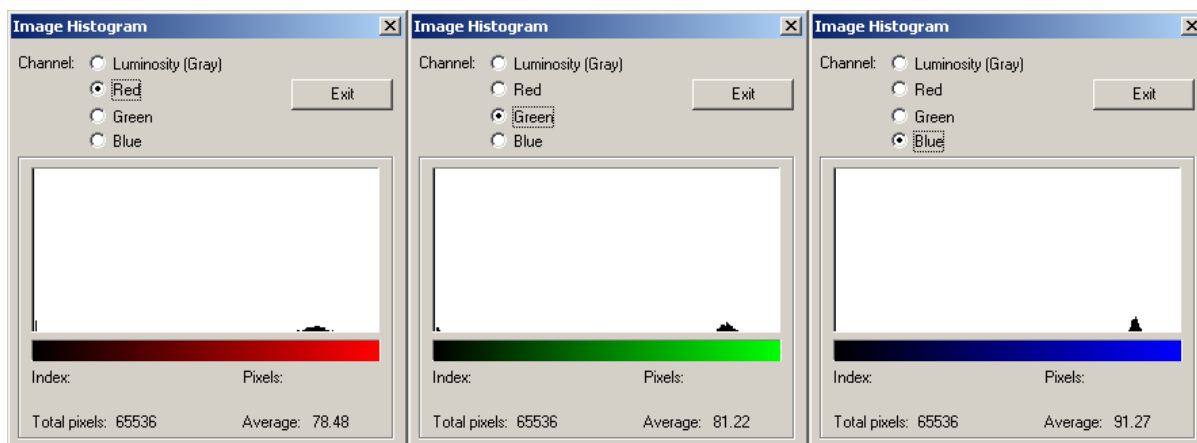
фигура 20 – Кодирано изображение

На фигура 19 е показано изображение, съдържащо буквата g, преди да бъде кодирано със скрито съобщение.

На фигура 20 е показано същото изображение, но с кодирано в него съобщение.



фигура 21 - Хистограма на разпределението на червен, зелен и син цвят в оригиналното изображение



фигура 22 - Хистограма на разпределението на червен, зелен и син цвят в кодираното изображение

На фигура 21 са показани хистограмите за червен, зелен и син цвят на оригиналното изображение, показано на фигура 19.

На фигура 22 са показани хистограмите на червен, зелен и син цвят на кодираното изображение, показано на фигура 20.

	Оригинално изображение	Кодирано изображение
Размер (Байтове)	39 660	40 860
Брой уникални цветове	5 437	5 981
Резолюция (пиксел x пиксел)	256 x 256	256 x 256
Дължина на паролата	-	4
Дължина на съобщението	-	8000
Има ли прозрачност?	Да	Да

таблица 18 – Сравнителни характеристики

6.5.Тест №5



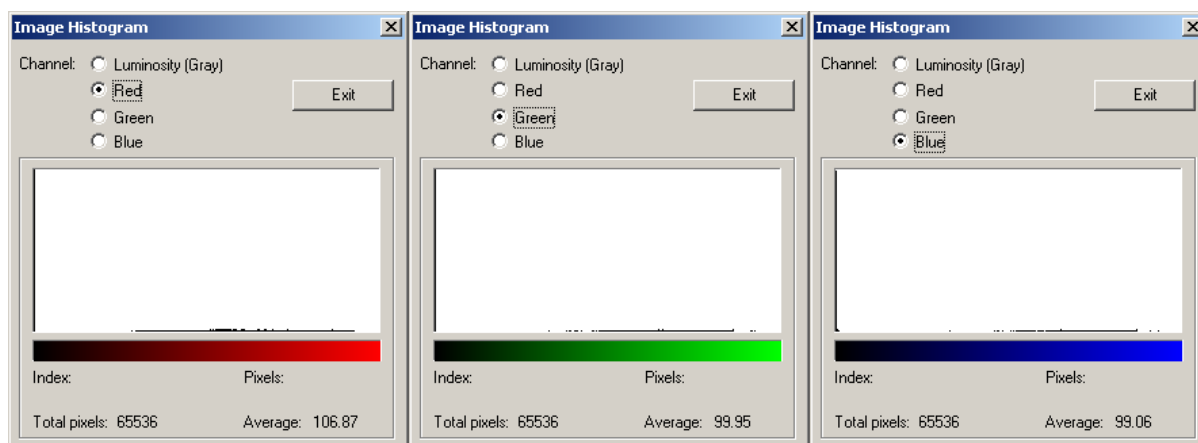
фигура 23 – Оригиналното изображение



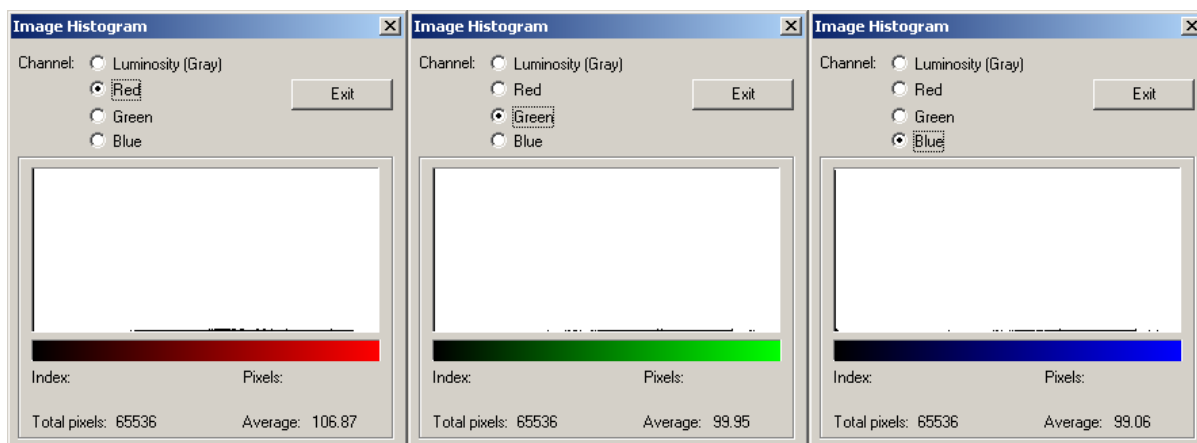
фигура 24 – Кодирано изображение

На фигура 23 е показано изображение, съдържащо популярния пингвин TUX, преди да бъде кодирано със скрито съобщение.

На фигура 24 е показано същото изображение, но с кодирано в него съобщение.



фигура 25 - Хистограма на разпределението на червен, зелен и син цвят в оригиналното изображение



фигура 26 - Хистограма на разпределението на червен, зелен и син цвят в кодираното изображение

На фигура 25 са показани хистограмите за червен, зелен и син цвят на оригиналното изображение, показано на фигура 23.

На фигура 26 са показани хистограмите на червен, зелен и син цвят на кодираното изображение, показано на фигура 24.

	Оригинално изображение	Кодирано изображение
Размер (Байтове)	108 013	111 199
Брой уникални цветове	11 997	13 018
Резолюция (пиксел x пиксел)	256 x 256	256 x 256
Дължина на паролата	-	4
Дължина на съобщението	-	8000
Има ли прозрачност?	Да	Да

таблица 19 – Сравнителни характеристики

7. Изводи

От направените опити могат да се направят няколко извода, сравнявайки изображенията на база видими разлики, хистограми и таблици:

- Колкото по-голямо е съобщението, което искаме да скрием, толкова по-голямо влияние оказва то върху разликите между двете изображения. Виждаме ясно, че примерите, в които изображението е с размер 8000 символа, това се отразява и върху размера, и върху уникалните цветове на кодираното изображение.
- При изображения с малки цветови разлики, например тези, които са черно бели, се забелязват повече разлики в сравнителните характеристики – в размера и в броя уникални цветове. Това се забелязва дори и при малък размер на съобщението. Колкото е цветовият диапазон е по-голям, толкова по малко разлики се забелязват. Това дава възможност в по-шарените изображения, да скрием по-големи съобщения.
- При кодиране на съобщения в изображения с алфа блендинг, прозрачността се запазва.
- При кодиране на съобщения, дори на такива с голям размер, хистограмите не търпят почти никакви промени.
- Резолуцията на кодираното изображение остава същата като на оригиналното. Това се дължи на факта, че промените се прилагат върху отделните пиксели.
- Дори при големи съобщения, разликата в големината на двете изображения е минимална. Това се дължи на факта, че при кодирането на битове има 50% възможност байтът да не бъде променен. Теоретично е възможно дори при кодиране на съобщение, крайният продукт да няма никаква промяна в размера.

8. Ръководство за потребителя

8.1. Начин на инсталация

Инсталацията на приложението е лесна. Трябва да се следват следните стъпки:

- Изтеглете *Java Development Kit 7* от следния линк
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>;
- Инсталирайте изтегления пакет;
- Конфигурирайте *bin* директорията от *java* да бъде в достъпна от всякъде. Това става като я добавите в променливата на ниво операционна система *\$PATH*. Обърнете внимание, че разделителят за пътища в тази променлива е различен за различните операционни системи - за Windows е „;“, а за Linux – „:“;
- Изтеглете *Apache Tomcat 7 Web* сървър от следния линк
<http://tomcat.apache.org/download-70.cgi>;
- Разархивирайте изтегления архив в директория по ваш избор;
- Ако *java runtime environment* е конфигуриран успешно, не би трябвало да имате проблеми за успешно стартиране на сървъра. Това става като изпълните ***startup.sh*** скрипта, намиращ се в ***bin*** директорията на *Tomcat*;
- Създайте *Java* интернет архив (*.war) файл. Това може да направите с подходящ инструмент като например *Eclipse* – зареждате проекта и избирате опция „Export“ или чрез *Ant* скрипт.
- Преместете току що направения архив в директорията ***webapps***, намираща се в папката на *Tomcat*;
- Създайте папка */tmp/steganography* и се убедете, че *Tomcat* сървърът има права да пише в нея. Тази папка се използва за съхранение на временните файлове на сървъра;
- Спрете и стартирайте отново сървъра, като използвате скриптовете ***shutdown.sh*** и ***startup.sh***;
- Отворете в интернет браузър адрес <http://localhost:8080/ImageSteganography/>

8.2. Начин на употреба

Приложението представлява интернет страница, на която можеш да кодираш и декодираш изображения. Всички *HTTP* заявки към сървъра минават през *AJAX*, а всички *HTTP* отговори – се обработват динамично с *javascript* в браузъра и променят съдържанието без да е нужно презареждане на цялата страница. Тази му функционалност прави приложението много

удобно за вграждане в други интернет приложения. Без никакъв проблем то може да бъде разположено в `<iframe>` таг и да стане част от нова страница.

Приложението се състои от два таба – един за кодиране (фигура 27) и един за декодиране (фигура 28) на изображения:

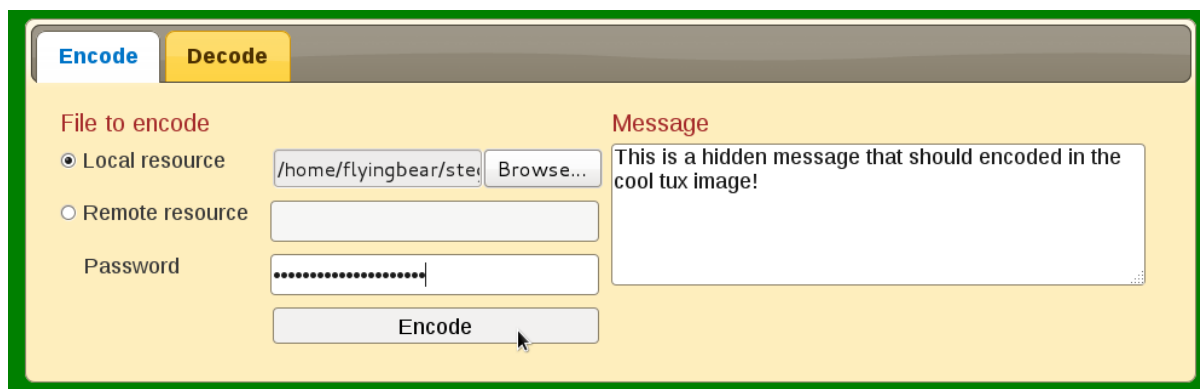
The screenshot shows the 'Encode' tab selected. At the top, there are two tabs: 'Encode' (active) and 'Decode'. Below the tabs, the interface is divided into two main sections. On the left, under the heading 'File to encode', there are three radio buttons: 'Local resource' (selected), 'Remote resource', and 'Password'. Each radio button is followed by a text input field. The 'Local resource' field has a 'Browse...' button next to it. Below these fields is an 'Encode' button. On the right, under the heading 'Message', there is a large text area for entering a message.

фигура 27 – Изглед на страницата за кодиране

The screenshot shows the 'Decode' tab selected. At the top, there are two tabs: 'Encode' and 'Decode' (active). Below the tabs, the interface is divided into two main sections. On the left, under the heading 'File to decode', there are three radio buttons: 'Local resource' (selected), 'Remote resource', and 'Password'. Each radio button is followed by a text input field. The 'Local resource' field has a 'Browse...' button next to it. Below these fields is a 'Decode' button. On the right, under the heading 'Message', there is a large text area for entering a message.

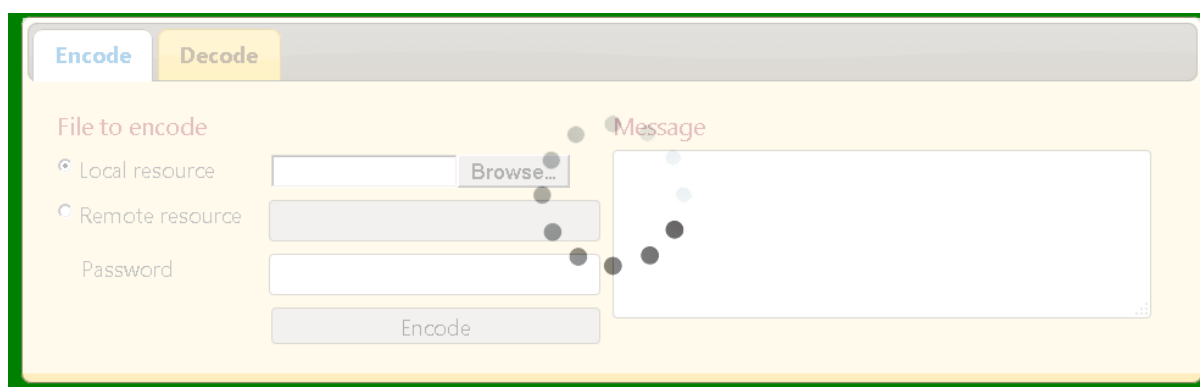
фигура 28 – Изглед на страницата за декодиране

За кодиране на изображение първо трябва да изберем изображение от локалната машина на клиента или да подадем интернет адрес за изображение, намиращо се на отдалечена машина. Това става като изберем една от опциите *Local resource* или *Remote resource*, реализирани с радио бутон. След този избор, едно от двете полета – за указване на файл, ще се активира, а другото – деактивира. Потребителят има възможност да въведе парола, от която зависи, кой ще е индексът на стартовия байт от изображението. Ако такава не бъде въведена, стартовият байт ще бъде с индекс 0. Последното поле, което трябва да бъде попълнено е полето за съобщението. Съобщението трябва да е до 9999 символа, поради причини, обяснени в една от предните глави. След това трябва да се натисне бутона *Encode* (фигура 29).



фигура 29 – Въвеждане на данни за кодиране

При натискане на бутона *Encode* се генерира AJAX заявка към сървъра. По време на изпълнението ѝ активният таб е закрит от изображение, индикиращо зареждане (фигура 30).



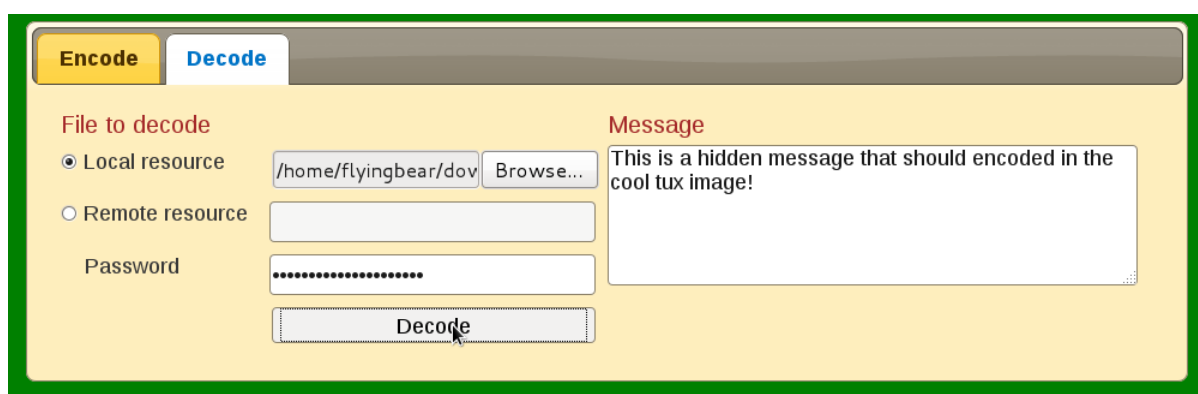
фигура 30 – Зареждане по време на кодиране

При успешно изпълнение, ще се визуализират двете изображения – оригиналното и кодираното, за да може потребителят да ги сравни визуално. При едно натискане с мишката върху изображението, то се отваря в нов таб на браузъра, в пълния си размер и потребителят има възможност да го изтегли локално на машината си (фигура 31).



фигура 31 – Резултат след кодиране на съобщение

За декодиране на изображение, стъпките са подобни. При избор на таба за декодиране, се визуализира интерфейс, подобен на този за кодиране. Потребителят трябва да въведе пътят до изображението или интернет адрес, ако то е разположено на отдалечен сървър. Също трябва да се въведе и парола, ако има такава. При несъответствие на паролата, приложението ще се опита да прочете грешни байтове от изображението и ще бъде върната грешка (фигура 32).



фигура 32 – Пример за декодиране

9. Заключение

Софтуерния продукт, разработен в тази дипломна работа има лесен и удобен потребителски интерфейс. Потребителят може лесно да кодира и декодира съобщения от изображения. Също интерфейсът може да бъде вграден в други програмни интернет решения поради AJAX комуникацията си със сървъра. Също заради архитектурата на кода си е лесно разширим и надградим.

Както се вижда от опитните резултати, LSB алгоритъма е лесен, удобен и функционален за скриване на съобщение в изображения. Разбира се приложението може да се разрастне, като се комбинира с други стеганографски и оптимизационни методи. Също може да бъде добавена функционалност за поддръжка на други изображения или на друг вид съобщения – не само текстови, но и двоични.

10. Програмен код

10.1. Steganography.java

```
package org.university.steganography.algorithm;

import java.awt.image.BufferedImage;

import org.university.steganography.exception.SteganographyException;
import org.university.steganography.util.Constants;
import org.university.steganography.util.Log;
import org.university.steganography.util.MessageConstants;

/**
 * Class used to encode and decode messages in image objects. <b>LSB (Least
 * Significant Bit)</b> algorithm is used.
 *
 * @author Kiril Aleksandrov
 */
public class Steganography
{
    /**
     * Encodes message into the given image
     *
     * @param bufferedImage
     *         The image that will carry the encoded message
     * @param message
     *         The message to encoded
     * @param offset
     *         The index of the starting byte
     * @return New image encoded with the message
     * @throws SteganographyException
     */
    public BufferedImage encode(final BufferedImage bufferedImage, final String
message, final int offset)
        throws SteganographyException
    {
        final BufferedImage image = this.addText(bufferedImage, message,
offset);
        return image;
    }

    /**
     * Decodes and gets a hidden message (if it exists) from the given image.
     * The used algorithm is <b>LSB (Least Significant Bit)</b>
     *
     * @param bufferedImage
     *         The image to be docoded
     * @param startingOffset
     *         The index of the starting byte
     * @return The message extracted from the image
     * @throws SteganographyException
     */
    public String decode(final BufferedImage bufferedImage, final int
startingOffset) throws SteganographyException
```

```

{
    byte[] decode = null;
    try
    {
        decode = this.decodeText(bufferedImage, startingOffset);
        final String decodedMessage = new String(decode);
        return decodedMessage;
    }
    catch (Exception e)
    {
        throw new
SteganographyException(MessageConstants.DECODING_ERROR_GENERAL);
    }
}

/**
 * Encodes text into the given image
 *
 * @param bufferedImage
 *         The image that will carry the encoded message
 * @param text
 *         The text to encoded
 * @param offset
 *         The index of the starting byte
 * @return New image encoded with the text
 * @throws SteganographyException
 */
private BufferedImage addText(BufferedImage image, final String text, final
int offset)
    throws SteganographyException
{
    // Convert all items to byte arrays: image, message, message length
    final byte msg[] = text.getBytes();
    final byte len[] = bitConversion(msg.length);
    try
    {
        // Encode the message length
        Log.debug("111111111111");
        image = this.encodeText(image, len, offset);
        Log.debug("222222222222");
        // Encode the message
        image = this.encodeText(image, msg, offset +
Constants.HIDDEN_MESSAGE_BIT_LENGTH);
        Log.debug("333333333333");
    }
    catch (Exception e)
    {
        throw new
SteganographyException(MessageConstants.ENCODING_ERROR_GENERAL, e);
    }

    return image;
}

/**
 * Converts the given number to array of bytes
 *
 * @param i
 *         The number to be converted

```

```

    * @return The byte array representing the number
    */
    private byte[] bitConversion(final int i)
    {
        final byte byte3 = (byte) ((i & 0xFF000000) >>> 24);
        final byte byte2 = (byte) ((i & 0x00FF0000) >>> 16);
        final byte byte1 = (byte) ((i & 0x0000FF00) >>> 8);
        final byte byte0 = (byte) ((i & 0x000000FF));
        return (new byte[]
        { byte3, byte2, byte1, byte0 });
    }

    /**
     * Encodes text into the given image
     *
     * @param bufferedImage
     *         The image that will carry the encoded message
     * @param addition
     *         The text to encoded
     * @param offset
     *         The index of the starting byte
     * @return New image encoded with the text
     * @throws SteganographyException
     */
    private BufferedImage encodeText(final BufferedImage image, final byte[]
    addition, final int offset)
        throws SteganographyException
    {
        // Gets the image dimensions
        final int height = image.getHeight();
        final int width = image.getWidth();

        // Initialize variables for iteration
        int i = offset / height;
        int j = offset % height;

        Log.info("Width : " + width);
        Log.info("Height : " + height);

        if ((width * height) >= (addition.length * 8 + offset))
        {
            // Iterates over all message's bytes
            for (final byte add : addition)
            {
                // Iterates over all of the bits in the current byte
                for (int bit = 7; bit >= 0; --bit)
                {
                    Log.info("[ " + i + " , " + j + " ]");
                    // Gets the original image byte value
                    final int imageValue = image.getRGB(i, j);

                    // Calculates the new image byte value
                    int b = (add >>> bit) & 1;
                    final int imageNewValue = ((imageValue &
0xFFFFFFFF) | b);

                    // Sets the new image byte value
                    image.setRGB(i, j, imageNewValue);
                }
            }
        }
    }

```

```

        if (j < (height - 1))
        {
            ++j;
        }
        else
        {
            ++i;
            j = 0;
        }
    }

    }
    else
    {
        throw new
SteganographyException(MessageConstants.ENCODING_ERROR_BIG_MESSAGE);
    }

    return image;
}

/**
 * Decodes and gets a hidden message (if it exists) from the given image.
 * The used algorithm is <b>LSB (Least Significant Bit)</b>
 *
 * @param image
 *         The image to be decoded
 * @param startingOffset
 *         The index of the starting byte
 * @return The message extracted from the image
 */
private byte[] decodeText(final BufferedImage image, final int
startingOffset)
{
    // Initialize starting variables
    final int width = image.getWidth();
    final int height = image.getHeight();
    final int offset = startingOffset +
Constants.HIDDEN_MESSAGE_BIT_LENGTH;
    int length = 0;

    // Loop through 32 bytes of data to determine text length
    for (int i = startingOffset; i < offset; ++i)
    {
        final int h = i / height;
        final int w = i % height;

        final int imageValue = image.getRGB(h, w);
        length = (length << 1) | (imageValue & 1);
    }

    Log.info("LENGTH : " + length);

    byte[] result = new byte[length];

    // Initialize variables for iteration
    int i = offset / height;
    int j = offset % height;

```

```

        // Iterate from zero to message length
        for (int letter = 0; letter < length; ++letter)
        {
            // Iterates over each bit for the hidden message
            for (int bit = 7; bit >= 0; --bit)
            {
                // Gets the byte from the image
                final int imageValue = image.getRGB(i, j);

                // Calculates the bit for the message
                result[letter] = (byte) ((result[letter] << 1) |
(imageValue & 1));

                if (j < (height - 1))
                {
                    ++j;
                }
                else
                {
                    ++i;
                    j = 0;
                }
            }
        }

        return result;
    }
}

```

10.2. ImageType.java

```

package org.university.steganography.enumeration;

/**
 * Enumeration representing known image types
 *
 * @author Kiril Aleksandrov
 */
public enum ImageType
{
    JPG, PNG
}

```

10.3. SteganographyException.java

```

package org.university.steganography.exception;

/**
 * Generic exception thrown if something wrong occurs during steganography
 * algorithm execution.
 */

```

```

*
* @author Kiril Aleksandrov
*
*/
public class SteganographyException extends Exception
{
    private static final long serialVersionUID = 9031373046571531684L;

    public SteganographyException()
    {
        super();
    }

    public SteganographyException(String message)
    {
        super(message);
    }

    public SteganographyException(Throwable throwable)
    {
        super(throwable);
    }

    public SteganographyException(String message, Throwable throwable)
    {
        super(message, throwable);
    }
}

```

10.4. DecodeServletResponse.java

```

package org.university.steganography.servlet.response;

/**
 * This class is used to represent the response from <b>DecodeImageServlet</b>
 *
 * @author Kiril Aleksandrov
 *
 */
public class DecodeServletResponse extends ServletResponse
{
    /**
     * The decoded message
     */
    private String message;

    public String getMessage()
    {
        return message;
    }

    public void setMessage(String message)
    {
        this.message = message;
    }
}

```


10.5. EncodeServletResponse.java

```
package org.university.steganography.servlet.response;

/**
 * This class is used to represent the response from the
 * <b>EncodeImageServlet</b>
 *
 * @author Kiril Aleksandrov
 */
public class EncodeServletResponse extends ServletResponse
{
}
```

10.6. Message.java

```
package org.university.steganography.servlet.response;

/**
 * Message that is returned from the server to the client. Possible types are
 * <b>INFO</b>, <b>WARN</b> and <b>ERROR</b>.
 *
 * @author Kiril Aleksandrov
 */
public class Message
{
    /**
     * The type of the message
     */
    private String type;

    /**
     * The text of the message
     */
    private String text;

    public Message()
    {
    }

    public Message(String type, String text)
    {
        this.type = type;
        this.text = text;
    }

    public String getType()
    {
        return type;
    }
}
```

```

    }

    public void setType(String type)
    {
        this.type = type;
    }

    public String getText()
    {
        return text;
    }

    public void setText(String text)
    {
        this.text = text;
    }
}

```

10.7. ServletResponse.java

```

package org.university.steganography.servlet.response;

/**
 * Abstract class representing the HTTP response from any servlet. All servlet
 * responses should extends this class.
 *
 * @author Kiril Aleksandrov
 */
public abstract class ServletResponse
{
    /**
     * The status of the performed operation
     */
    private Status status;

    public Status getStatus()
    {
        return status;
    }

    public void setStatus(Status status)
    {
        this.status = status;
    }
}

```

10.8. Status.java

```

package org.university.steganography.servlet.response;

import java.util.ArrayList;
import java.util.List;

```

```

import org.university.steganography.util.Constants;

/**
 * The status of the performed operation. If something wrong occured the field
 * <b>success</b> should be false.
 *
 * @author Kiril Aleksandrov
 */
public class Status
{
    /**
     * Flag indicating if the performed operation was successfully
     */
    private boolean success;

    /**
     * List of messages that are sent from the server to the client
     */
    private List<Message> messages;

    public Status()
    {
        this.success = true;
        this.messages = new ArrayList<Message>();
    }

    public boolean isSuccess()
    {
        return success;
    }

    /**
     * Add message to the list of messages. If the message type equals to
     * <b>ERROR</b> the <b>success</b> flag should be turned to false
     *
     * @param type
     *         The type of the message
     * @param text
     *         The message content
     */
    public void addMessage(String type, String text)
    {
        if (Constants.MESSAGE_TYPE_ERROR.equalsIgnoreCase(type) &&
this.success)
        {
            this.success = false;
        }

        Message message = new Message(type, text);
        this.messages.add(message);
    }

    public List<Message> getMessages()
    {
        return this.messages;
    }
}

```

10.9. DecodeImageServlet.java

```
package org.university.steganography.servlet;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;

import org.university.steganography.algorithm.Steganography;
import org.university.steganography.exception.SteganographyException;
import org.university.steganography.servlet.response.DecodeServletResponse;
import org.university.steganography.servlet.response.Status;
import org.university.steganography.util.Constants;
import org.university.steganography.util.Log;
import org.university.steganography.util.Utils;

import com.google.gson.Gson;

/**
 * The instances of this class are server(context) managed. Only HTTP POST
 * requests are allowed The servlet decodes the image sent through the HTTP
 * request. Returns the decoded message in the response object along with the
 * status object containing server messages. The response is serialized az JSON
 * (JavaScript Object Nation) object
 *
 * @author Kiril Aleksandrov
 */
@WebServlet(urlPatterns =
{ Constants.SERVLET_DECODE_IMAGE })
@MultipartConfig(location = Constants.CONF_TEMP_DIRECTORY, maxFileSize =
Constants.CONF_MAX_FILE_SIZE)
public class DecodeImageServlet extends HttpServlet
{
    private static final long serialVersionUID = 1L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    {
        // Initialize variables and ger HTTP request parameters

        Status status = new Status();

        String sourceType =
request.getParameter(Constants.PARAM_DECODED_SOURCE);
```

```

        String password =
request.getParameter(Constants.PARAM_DECODE_PASSWORD);
        BufferedImage image = null;

        try
        {
            if (Constants.SOURCE_TYPE_LOCAL.equalsIgnoreCase(sourceType))
            {
                // Read the image from the request object
                Part filePart =
request.getPart(Constants.PARAM_FILE_DECODE_LOCAL);

                InputStream filecontent = filePart.getInputStream();
                image = Utils.streamToImage(filecontent);
                filecontent.close();
            }
            else if
(Constants.SOURCE_TYPE_REMOTE.equalsIgnoreCase(sourceType))
            {
                // Read the request from remote resource
                String remoteAddress =
request.getParameter(Constants.PARAM_FILE_DECODE_REMOTE);
                image = Utils.loadRemoteFile(remoteAddress);
            }
        }
        catch (IOException | ServletException e)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error
uploading image");
        }

        final int imageLength = image.getHeight() * image.getWidth();
        final int startingOffset = Utils.calculateStartingOffset(password,
imageLength);
        Log.info("Starting byte index : " + startingOffset);

        // Decode the image
        Steganography steganography = new Steganography();
        String message = null;
        try
        {
            message = steganography.decode(image, startingOffset);
        }
        catch (SteganographyException e1)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error
decoding image");
        }

        request.setAttribute(Constants.ATTR_MESSAGE, message);

        DecodeServletResponse responseWrapper = new DecodeServletResponse();
        responseWrapper.setStatus(status);

        responseWrapper.setMessage(message);

        // Writes the response object to the HTTP response stream that is
        // returned to the client
        PrintWriter out = null;

```

```

        try
        {
            out = new PrintWriter(response.getOutputStream());
        }
        catch (IOException e)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error writing
response");
        }

        if (null != out)
        {
            Gson gson = new Gson();
            Log.info(gson.toJson(responseWrapper));

            out.write(gson.toJson(responseWrapper));

            out.close();
            out.flush();
        }
    }
}

```

10.10. EncodeImageServlet.java

```

package org.university.steganography.servlet;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.Part;

import org.university.steganography.algorithm.Steganography;
import org.university.steganography.exception.SteganographyException;
import org.university.steganography.servlet.response.EncodeServletResponse;
import org.university.steganography.servlet.response.Status;
import org.university.steganography.util.Constants;
import org.university.steganography.util.Log;
import org.university.steganography.util.Utills;

import com.google.gson.Gson;

/**
 * The instances of this class are server(context) managed. Only HTTP POST
 * requests are allowed The servlet encodes the image with the message both sent
 * through the HTTP request. The response is serialized az JSON (JavaScript
 * Object Nation) object
 */

```

```

* @author Kiril Aleksandrov
*
*/
@WebServlet(urlPatterns =
{ Constants.SERVLET_ENCODE_IMAGE })
@MultipartConfig(location = Constants.CONF_TEMP_DIRECTORY, maxFileSize =
Constants.CONF_MAX_FILE_SIZE)
public class EncodeImageServlet extends HttpServlet
{
    private static final long serialVersionUID = 1L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
        IOException
    {
        // Initialize variables and gets HTTP request parameters
        Status status = new Status();

        String message =
request.getParameter(Constants.PARAM_HIDDEN_MESSAGE);
        String password =
request.getParameter(Constants.PARAM_ENCODE_PASSWORD);

        BufferedImage image = null;

        String sourceType =
request.getParameter(Constants.PARAM_ENCODED_SOURCE);

        try
        {
            if ("local".equalsIgnoreCase(sourceType))
            {
                // Loads the image from the HTTP request object
                Part filePart =
request.getPart(Constants.PARAM_FILE_ENCODE_LOCAL);
                InputStream fileInputStream = filePart.getInputStream();
                image = Utils.streamToImage(fileInputStream);
                fileInputStream.close();
            }
            else if ("remote".equalsIgnoreCase(sourceType))
            {
                // Loads the image from remote resource
                String remoteAddress =
request.getParameter(Constants.PARAM_FILE_ENCODE_REMOTE);
                image = Utils.LoadRemoteFile(remoteAddress);
            }
        }
        catch (IOException | ServletException e)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error
uploading image");
        }

        // Encodes the message in the image
        Log.info("Image : " + (image != null));
        final int imageLength = image.getHeight() * image.getWidth();
        Log.info("Image length : " + imageLength);
    }
}

```

```

        final int startingOffset = Utils.calculateStartingOffset(password,
imageLength);
        Log.info("Starting byte index : " + startingOffset);

        Steganography steganography = new Steganography();
        BufferedImage encodedImage = null;
        try
        {
            encodedImage = steganography.encode(image, message,
startingOffset);
        }
        catch (SteganographyException e)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error
encoding image");
        }

        // Sets the original and the encoded image to the user session
object.
        // Thus they are available for later use.
        HttpSession session = request.getSession();
        synchronized (session)
        {
            session.setAttribute(Constants.ATTR_ORIGINAL_IMAGE, image);
            session.setAttribute(Constants.ATTR_ENCODED_IMAGE,
encodedImage);
        }

        EncodeServletResponse responseWrapper = new EncodeServletResponse();

        responseWrapper.setStatus(status);

        // Writes the response object to the HTTP response stream
        PrintWriter out = null;
        try
        {
            out = new PrintWriter(response.getOutputStream());
        }
        catch (IOException e)
        {
            status.addMessage(Constants.MESSAGE_TYPE_ERROR, "Error writing
response");
        }

        if (null != out)
        {
            Gson gson = new Gson();
            Log.info(gson.toJson(responseWrapper));

            out.write(gson.toJson(responseWrapper));

            out.close();
            out.flush();
        }
    }
}

```


10.11. ImageServlet.java

```
package org.university.steganography.servlet;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;

import javax.imageio.ImageIO;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.university.steganography.enumeration.ImageType;
import org.university.steganography.util.Constants;

/**
 * The instances of this class are server(context) managed. Only HTTP GET
 * requests are allowed The servlet returns an image from user's session. The
 * image can be the original image that the user has uploaded or the encoded
 * image thath has been encoded with the message. This is controlled by the HTTP
 * request parameter <b>version</b>.
 *
 * @author Kiril Aleksandrov
 */
@WebServlet(urlPatterns =
{ Constants.SERVLET_IMAGE })
public class ImageServlet extends HttpServlet
{
    private static final long serialVersionUID = -7780986917524240245L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
    {
        // Set response content type and file properties
        response.setContentType("image/png");
        response.setHeader("Content-Disposition", "filename=image");

        String imageVersion =
request.getParameter(Constants.PARAM_IMAGE_VERSION);

        HttpSession session = request.getSession();

        BufferedImage image = null;

        // Gets the image from the session according to the request
parameter
        // 'version'
        synchronized (session)
        {
            if
(Constants.PARAM_IMAGE_VERSION_ORIGINAL.equals(imageVersion))
```

```

        {
            image = (BufferedImage)
session.getAttribute(Constants.ATTR_ORIGINAL_IMAGE);
        }
        else if
(Constants.PARAM_IMAGE_VERSION_ENCODED.equals(imageVersion))
        {
            image = (BufferedImage)
session.getAttribute(Constants.ATTR_ENCODED_IMAGE);
        }
    }

    // Add the image to the HTTP resposne stream so it can be loaded by
the
    // browser
    if (null != image)
    {
        ImageType imageType = ImageType.PNG;
        OutputStream out = response.getOutputStream();
        ImageIO.write(image, imageType.toString(), out);
        out.close();
        out.flush();
    }
}
}

```

10.12. Constants.java

```

package org.university.steganography.util;

/**
 * Public interface containing all of the constants needed for the application
 *
 * @author Kiril Aleksandrov
 *
 */
public interface Constants
{
    // Common

    String EMPTY_STRING = "";

    // Servlet mappings

    String SERVLET_ENCODE_IMAGE = "/encode";

    String SERVLET_DECODE_IMAGE = "/decode";

    String SERVLET_IMAGE = "/image";

    // Configuration

    String CONF_TEMP_DIRECTORY = "/home/flyingbear/temp";

    long CONF_MAX_FILE_SIZE = 20971520L; // 20 MB

```

```

// Test

String TEST_MESSAGE = "This is a test";

String TEST_IMAGE = "/home/flyingbear/stego/test_";

// Parameters

String PARAM_FILE_ENCODE_LOCAL = "file-encode-local";

String PARAM_FILE_ENCODE_REMOTE = "file-encode-remote";

String PARAM_FILE_DECODE_LOCAL = "file-decode-local";

String PARAM_FILE_DECODE_REMOTE = "file-decode-remote";

String PARAM_IMG_TYPE = "image-type";

String PARAM_IMG_TYPE_PNG = "png";

String PARAM_IMG_TYPE_JPG = "jpg";

String PARAM_HIDDEN_MESSAGE = "hidden-message";

String PARAM_IMAGE_VERSION = "version";

String PARAM_IMAGE_VERSION_ORIGINAL = "original";

String PARAM_IMAGE_VERSION_ENCODED = "encoded";

String PARAM_ENCODED_SOURCE = "encode-source";

String PARAM_DECODED_SOURCE = "decode-source";

String PARAM_ENCODE_PASSWORD = "input-encode-password";

String PARAM_DECODE_PASSWORD = "input-decode-password";

// Attributes

String ATTR_ORIGINAL_IMAGE = "originalImage";

String ATTR_ENCODED_IMAGE = "encodedImage";

String ATTR_IMAGE_TYPE = "encodedImageType";

String ATTR_MESSAGE = "decodedMessage";

String ATTR_ENCODE_MESSAGE = "encodeMessage";

String ATTR_ENCODE_FILE_TYPE = "encodeFileType";

// Source types

String SOURCE_TYPE_LOCAL = "local";

String SOURCE_TYPE_REMOTE = "remote";

// Message types

```

```

String MESSAGE_TYPE_ERROR = "error";

String MESSAGE_TYPE_INFO = "info";

String MESSAGE_TYPE_WARNING = "warning";

// Message byte length

int HIDDEN_MESSAGE_BIT_LENGTH = 32;
}

```

10.13. ImageUtils.java

```

package org.university.steganography.util;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.PixelGrabber;
import java.io.File;

import javax.activation.MimetypesFileTypeMap;
import javax.servlet.ServletException;

import org.university.steganography.enumeration.ImageType;

/**
 * Utility class holding methods for image processing
 *
 * @author Kiril Aleksandrov
 */
public class ImageUtils
{
    private ImageUtils()
    {
        // Prevent initialization
    }

    /**
     * Check if the given message has <b>alpha</b> channel pixels
     *
     * @param image
     *         The image to be inspected for alpha channel
     * @return <b>true</b> if the image contains alpha channel pixels,
otherwise
     *         - <b>false</b>
     */
    public static boolean hasAlpha(Image image)
    {
        // If buffered image, the color model is readily available
        if (image instanceof BufferedImage)
        {
            BufferedImage bimage = (BufferedImage) image;
            return bimage.getColorModel().hasAlpha();

```

```

    }

    // Use a pixel grabber to retrieve the image's color model;
    // grabbing a single pixel is usually sufficient
    PixelGrabber pg = new PixelGrabber(image, 0, 0, 1, 1, false);
    try
    {
        pg.grabPixels();
    }
    catch (InterruptedException e)
    {
    }

    // Get the image's color model
    ColorModel cm = pg.getColorModel();
    return cm.hasAlpha();
}

/**
 * Gets the file MIME type
 *
 * @param image
 *         The file to be inspected
 * @return The file type
 */
public static ImageType getImageMimeType(final File image)
{
    final MimeTypeMap mimeTypeMap = new MimeTypeMap();
    final String mimeType = mimeTypeMap.getContentType(image);

    ImageType imageType = null;

    if (ImageType.JPG.equals(mimeType))
    {
        imageType = ImageType.JPG;
    }
    else if (ImageType.PNG.equals(mimeType))
    {
        imageType = ImageType.PNG;
    }

    return imageType;
}

/**
 * Converts <b>String</b> value to value of <b>ImageType</b> enumeration.
 *
 * @param imageTypeStr
 *         Image type as String
 * @return Image type as enumeration value
 * @throws ServletException
 */
public static ImageType getImageType(String imageTypeStr) throws
ServletException
{
    ImageType imageType = null;

    if (Constants.PARAM_IMG_TYPE_JPG.equals(imageTypeStr))
    {

```

```

        imageType = ImageType.JPG;
    }
    else if (Constants.PARAM_IMG_TYPE_PNG.equals(imageTypeStr))
    {
        imageType = ImageType.PNG;
    }
    else
    {
        throw new ServletException("Not supported media type!");
    }

    return imageType;
}
}

```

10.14. Log.java

```

package org.university.steganography.util;

/**
 * Utility class to log useful information
 *
 * @author Kiril Aleksandrov
 *
 */
public class Log
{
    /**
     * Log informational message
     *
     * @param message
     */
    public static void info(final String message)
    {
        System.out.println(message);
    }

    /**
     * Log debug message
     *
     * @param message
     */
    public static void debug(final String message)
    {
        System.out.println(message);
    }

    /**
     * Log warning message
     *
     * @param message
     */
    public static void warn(final String message)
    {
        System.out.println(message);
    }
}

```

```

    }

    /**
     * Log error message
     *
     * @param message
     */
    public static void error(final String message)
    {
        System.out.println(message);
    }

    /**
     * Log fatal message
     *
     * @param message
     */
    public static void fatal(final String message)
    {
        System.out.println(message);
    }
}

```

10.15. MessageConstants.java

```

package org.university.steganography.util;

/**
 * Public interface holding message constants
 *
 * @author Kiril Aleksandrov
 *
 */
public interface MessageConstants
{
    /**
     * Encoding
     */
    /** Error

    String ENCODING_ERROR_GENERAL = "Error encoding image";

    String ENCODING_ERROR_BIG_MESSAGE = "Too big message";

    /**
     * Decoding
     */
    /** Error

    String DECODING_ERROR_GENERAL = "Error decoding image";
}

```

10.16. Utils.java

```
package org.university.steganography.util;

import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import java.awt.image.WritableRaster;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;

import javax.imageio.ImageIO;
import javax.servlet.http.Part;

/**
 * Public utility class holding a lot of useful methods
 *
 * @author Kiril Aleksandrov
 *
 */
public class Utils
{
    /**
     * Private constructor preventing outer initialization
     */
    private Utils()
    {
        // Prevent initialization
    }

    /**
     * Checks if the given String is empty. A String is considered as empty
only
     * if it is equal to <b>null</b> or it is has length of <b>0</b>
     *
     * @param value
     * @return <b>true</b> if the String is empty, otherwise - <b>false</b>
     */
    public static boolean isEmpty(final String value)
    {
        return (null == value ||
Constants.EMPTY_STRING.equalsIgnoreCase(value.trim()));
    }

    /**
     * Gets an image from the given stream
     *
     * @param stream
     *      Stream to get the image from
     * @return The extracted image
     * @throws IOException
     */
    public static BufferedImage streamToImage(InputStream stream) throws
IOException
    {
        BufferedImage image = ImageIO.read(stream);
        return image;
    }
}
```



```

    }

    /**
     * Gets the filename of the uploaded file
     *
     * @param part
     *         The uploaded file
     * @return The filename
     */
    public static String getFilename(Part part)
    {
        for (String cd : part.getHeader("content-disposition").split(";"))
        {
            if (cd.trim().startsWith("filename"))
            {
                String filename = cd.substring(cd.indexOf('=') +
1).trim().replace("\"", "");
                return filename.substring(filename.lastIndexOf('/') +
1).substring(filename.lastIndexOf('\\') + 1); // MSIE

                // fix.
            }
        }
        return null;
    }

    /**
     * Convertes the given image to array of bytes
     *
     * @param image
     *         Image to be converted
     * @return Array of bytes
     */
    public static byte[] getImageAsBytes(BufferedImage image)
    {
        WritableRaster raster = image.getRaster();
        DataBufferByte buffer = (DataBufferByte) raster.getDataBuffer();
        return buffer.getData();
    }

    /**
     * Compares two images byte by byte
     *
     * @param original
     *         The original image
     * @param transformed
     *         The transformed image
     */
    public static void compareImages(BufferedImage original, BufferedImage
transformed)
    {
        byte[] originalBytes = Utils.getImageAsBytes(original);
        byte[] transformedBytes = Utils.getImageAsBytes(transformed);

        Log.info("Original bytes : " + originalBytes.length);
        Log.info("Transformed bytes : " + transformedBytes.length);
    }

```

```

        long length = Math.min(originalBytes.length,
transformedBytes.length);

        boolean different = false;

        for (int i = 0; i < length; i++)
        {
            if (originalBytes[i] != transformedBytes[i])
            {
                different = true;
            }
        }

        Log.info("DIFFERENT : " + different);
    }

    /**
     * Loads image from remote internet address
     *
     * @param remoteAddress
     *         Remote internet address
     * @return The loaded image
     * @throws IOException
     */
    public static BufferedImage loadRemoteFile(String remoteAddress) throws
IOException
    {
        final URL url = new URL(remoteAddress);
        BufferedImage image = ImageIO.read(url);
        return image;
    }

    /**
     * Calculates the index of the starting byte
     *
     * @param password
     *         The password given from the user
     * @param maxValue
     *         The length of the image
     * @return The index of the starting byte
     */
    public static int calculateStartingOffset(final String password, final long
maxValue)
    {
        int offset = 0;

        Log.info("PASSWORD : " + password);
        Log.info("MAX_VALUE : " + maxValue);

        if (!Utils.isEmpty(password))
        {
            for (char c : password.toCharArray())
            {
                offset += c;
            }

            offset %= maxValue;
        }
    }

```

```

        return offset;
    }
}

```

10.17. Index.jsp

```

<!DOCTYPE html>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>

    <head>

        <link rel="shortcut icon"
href="${pageContext.request.contextPath}/theme/img/favicon.jpg" />
        <link type="text/css" rel="stylesheet"
href="${pageContext.request.contextPath}/theme/css/style.css" />
        <link type="text/css" rel="stylesheet"
href="${pageContext.request.contextPath}/theme/css/sunny/jquery-ui-
1.8.19.custom.css" />

        <script type="text/javascript"
src="${pageContext.request.contextPath}/theme/js/jquery-1.7.2.min.js"></script>
        <script type="text/javascript"
src="${pageContext.request.contextPath}/theme/js/jquery-ui-
1.8.19.custom.min.js"></script>
        <script type="text/javascript"
src="${pageContext.request.contextPath}/theme/js/jquery.form.js"></script>
        <script type="text/javascript"
src="${pageContext.request.contextPath}/theme/js/utls.js"></script>

        <title>Steganography</title>

    </head>

    <body>

        <div id="container">

            <div id="tabs">

                <ul>
                    <li><a href="#tab-encode">Encode</a></li>
                    <li><a href="#tab-decode">Decode</a></li>
                </ul>

                <div id="tab-encode">

                    <form
                        id="form-encode"

                        action="${pageContext.request.contextPath}/encode"
                        method="post"
                        enctype="multipart/form-data"
                        class="form">

```

```

class="messages"></div>

encode</div>

resource-Local "

        onchange="toggleEncodeResourceSelection()">

resource-Local "

Local "

resource-remote"

        onchange="toggleEncodeResourceSelection()">

resource-remote"

remote"

<div id="encode-messages"

<div class="left half">

    <div class="header">File to

    <div class="inner-container">

        <input
            type="radio"
            id="encode-radio-

            name="encode-source"
            class="left radio"
            value="local"

        <label
            for="encode-radio-

            class="label">
                Local resource
        </label>
        <input
            type="file"
            id="file-encode-local"
            name="file-encode-

            class="input"
            size="14" />
        <div class="clear"></div>

        <input
            type="radio"
            id="encode-radio-

            name="encode-source"
            class="left radio"
            value="remote"

        <label
            for="encode-radio-

            class="label">
                Remote resource
        </label>
        <input
            type="text"
            id="file-encode-remote"
            name="file-encode-

            class="input"/>
        <div class="clear"></div>

        <label

```

```

password"
20px;"

password"
password"

for="input-encode-
style="margin-left:
class="label">
Password
</label>
<input
type="password"
id="input-encode-
name="input-encode-
autocomplete="off"
class="input"/>
<div class="clear"></div>

<input
type="submit"
id="button-encode"
value="Encode"
class="input"

onclick="startLoadingEncode();" />

<div class="clear"></div>

</div>

</div>

<div class="right half" >

<div class="header">Message</div>

<div class="inner-container">

<textarea
id="hidden-message"
name="hidden-message"
class="textarea"
maxlength="9999"
rows="5"
cols="37">

${requestScope.encodeMessage }

</textarea>
<div class="clear"></div>

</div>

</div>

<div class="clear"></div>

</form>

<div class="clear"></div>

<div id="images" class="images">

```

```

<div class="left half">

    <div class="inner-container-left">

        <div class="header">Original
image</div>

        <div class="spacer"></div>
        <div class="clear"></div>

        <a
href="${pageContext.request.contextPath}/image?version=original" target="_blank"
>

        </a>

    </div>

</div>

<div class="right half">

    <div class="inner-container-right">

        <div class="header">Encoded
image</div>

        <div class="spacer"></div>
        <div class="clear"></div>

        <a
href="${pageContext.request.contextPath}/image?version=encoded" target="_blank">

        </a>

    </div>

</div>

```

```

        <div class="clear"></div>

    </div>

    <div id="Loading-encode" class="loading"></div>

</div>

<div id="tab-decode">

    <form
        id="form-decode"

        action="{pageContext.request.contextPath}/decode"
        method="post"
        enctype="multipart/form-data"
        class="form">

        <div id="decode-messages"

class="messages"></div>

        <div class="left half">

            <div class="header">File to

decode</div>

            <div class="inner-container">

                <input
                    type="radio"
                    id="decode-radio-

resource-Local "

                    name="decode-source"
                    class="left radio"
                    value="local "

                    onchange="toggleDecodeResourceSeLection()">

                <label
                    for="decode-radio-

resource-Local "

                    class="label ">
                    Local resource
                </label>
                <input
                    type="file"
                    id="file-decode-local "
                    name="file-decode-

Local "

                    class="input"
                    size="14"/>
                <div class="clear"></div>

                <input
                    type="radio"
                    id="decode-radio-

resource-remote "

                    name="decode-source"
                    class="left radio"
                    value="remote"

```

```

        onchange="toggleDecodeResourceSelection()">
resource-remote"

remote"

password"
20px;"

password"
password"

        <label
            for="decode-radio-

            class="label">
                Remote resource
        </label>
        <input
            type="text"
            id="file-decode-remote"
            name="file-decode-

            class="input"/>
        <div class="clear"></div>

        <label
            for="input-decode-

            style="margin-left:

            class="label">
                Password
        </label>
        <input
            type="password"
            id="input-decode-

            name="input-decode-

            autocomplete="off"
            class="input"/>
        <div class="clear"></div>

        <input
            type="submit"
            id="button-decode"
            value="Decode"
            class="input"

        onclick="startLoadingDecode();" />

        <div class="clear"></div>

    </div>

</div>

<div class="right half" >

    <div class="header">Message</div>

    <div class="inner-container">

        <textarea
            id="decoded-message"
            rows="5"
            cols="37"
            readonly="readonly"
            class="textarea">

```



```

    ${requestScope.decodedMessage}

    </textarea>
    <div class="clear"></div>

    </div>

    </div>

    </form>

    <div class="clear"></div>

    <div id="Loading-decode" class="Loading"></div>

    </div>

    </div>

    </body>

</html>

```

10.18. Utils.js

```

$(document).ready(
    function() {
        // Create jQueryUI tabs
        createTabs();

        // Attach ajax form submission
        bindFormForAjax('form-encode', stopLoadingEncode, updateImages,
            null);
        bindFormForAjax('form-decode', stopLoadingDecode,
            updateResultMessage, null);

        // Set default values
        $('#encode-radio-resource-local').attr('checked', 'checked');
        $('#file-encode-remote').attr('disabled', 'disabled');
        $('#hidden-message').val('');

        $('#decode-radio-resource-local').attr('checked', 'checked');
        $('#file-decode-remote').attr('disabled', 'disabled');
        $('#decoded-message').val('');
    });

$.urlParam = function(name) {
    var results = new RegExp('[\\?&]' + name + '=(^&#)*')
        .exec(window.location.href);
    if (!results) {
        return 0;
    }
    return results[1] || 0;
};

```

```

function createButtons() {
    $('#button-encode').button();
    $('#button-decode').button();
}

function bindFormForAjax(formId, completeCallback, successCallback,
    errorCallback) {
    $('#' + formId).ajaxForm({
        dataType : 'json',
        success : function(data) {
            if (null != successCallback) {
                successCallback(data);
            }
        },
        error : function(data) {
            if (null != errorCallback) {
                errorCallback(data);
            }
        },
        complete : function(data) {
            if (null != completeCallback) {
                completeCallback(data);
            }
        }
    });
}

function updateResultMessage(data) {
    console.log(data);
    console.log(data.status.success);

    jDecodeMessage = $('#decoded-message');
    if (true == data.status.success) {
        console.log(data.message);
        jDecodeMessage.val(data.message);
    } else {
        jDecodeMessage.val('');
    }

    var messagesList = '';
    if (data.status.messages.length > 0) {
        messagesList = '<ul>';
        $.each(data.status.messages, function(index, message) {
            messagesList += '<li class="' + message.type + '>' +
message.text
                                + '</li>';
        });
        messagesList += '</ul>';
    }

    $('#decode-messages').html(messagesList);
}

function updateImages(data) {
    console.log(data);

    if (true == data.status.success) {
        var originalImage = $("#image-original");
        var encodedImage = $("#image-encoded");
    }
}

```

```

        d = new Date();

        var originalSrc = originalImage.attr('src') + '&' + d.getTime();
        var encodedSrc = encodedImage.attr('src') + '&' + d.getTime();

        originalImage.attr('src', '');
        encodedImage.attr('src', '');

        $('#images').show();

        originalImage.attr("src", originalSrc);
        encodedImage.attr("src", encodedSrc);
    }

    var messagesList = '';
    if (data.status.messages.length > 0) {
        messagesList = '<ul>';
        $.each(data.status.messages, function(index, message) {
            messagesList += '<li class="' + message.type + '">' +
message.text
                                + '</li>';
        });
        messagesList += '</ul>';
    }

    $('#encode-messages').html(messagesList);
}

function createTabs() {
    $('#tabs').tabs();
}

function startLoadingEncode() {
    var loading = $('#loading-encode');
    var height = $('#tabs').height();
    loading.height(height);
    loading.show();
}

function startLoadingDecode() {
    var loading = $('#loading-decode');
    var height = $('#tabs').height();
    loading.height(height);
    loading.show();
}

function stopLoadingEncode() {
    $('#loading-encode').hide();
}

function stopLoadingDecode() {
    $('#loading-decode').hide();
}

function toggleEncodeResourceSelection() {
    var radioLocal = $('#encode-radio-resource-local');
    var radioRemote = $('#encode-radio-resource-remote');
    var inputLocal = $('#file-encode-local');

```

```

    var inputRemote = $('#file-encode-remote');

    if ('checked' == radioLocal.attr('checked')) {
        inputLocal.removeAttr('disabled');
        inputRemote.attr('disabled', 'disabled');
        inputRemote.val('');
    }
    if ('checked' == radioRemote.attr('checked')) {
        inputLocal.attr('disabled', 'disabled');
        inputLocal.val('');
        inputRemote.removeAttr('disabled');
    }
}

function toggleDecodeResourceSelection() {
    var radioLocal = $('#decode-radio-resource-local');
    var radioRemote = $('#decode-radio-resource-remote');
    var inputLocal = $('#file-decode-local');
    var inputRemote = $('#file-decode-remote');

    if ('checked' == radioLocal.attr('checked')) {
        inputLocal.removeAttr('disabled');
        inputRemote.attr('disabled', 'disabled');
        inputRemote.val('');
    }
    if ('checked' == radioRemote.attr('checked')) {
        inputLocal.attr('disabled', 'disabled');
        inputLocal.val('');
        inputRemote.removeAttr('disabled');
    }
}

```

10.19. Style.css

```

/*****
*      Main layout      *
*****/
body {
    background: green;
}

input[type='text'],input[type='password'],input[type='submit'],textarea
{
    border: 1px solid #8E846B;
    border-radius: 4px;
}

input[type='text'],input[type='password'],input[type='submit'] {
    height: 30px;
}

#container {
    width: 960px;
    margin: auto;
}

```

```

#tabs {
    width: 100%;
}

/*****
*       File upload and message       *
*****/
.form {
    width: 100%;
}

.form .inner-container {
    margin: 5px 5px 5px 0;
}

.radio {
    margin: 8px 0;
}

.label,.input {
    margin: 5px;
}

.label {
    float: left;
    width: 140px;
}

.input {
    float: right;
    width: 270px;
}

.textarea {
    width: 440px;
}

.header {
    color: brown;
    font-size: 20px;
    margin: 5px 0;
}

.decoded-message {
    background-color: #EEEEEE;
    border: 1px solid;
    color: #000000;
}

/*****
*       Images       *
*****/
.image {
    max-width: 440px;
    max-height: 1200px;
}

.images {
    margin: 20px 0 0 0;
}

```

```

        width: 100%;
        display: none;
    }

    .images .image-container {
        width: 50%;
    }

    .images .inner-container-right {
        margin: 0 0 0 10px;
        float: left;
    }

    .images .inner-container-left {
        margin: 0 10px 0 0;
        float: right;
    }

    /*****
    *           Messages           *
    *****/
    .messages {

    }

    .messages .error {
        color: red;
    }

    .messages .info {
        color: green;
    }

    .messages .warn {
        color: orange;
    }

    /*****
    *           Common           *
    *****/
    .half {
        width: 50%;
    }

    .spacer {
        margin: 10px;
    }

    .clear {
        clear: both;
    }

    .left {
        float: left;
    }

    .right {
        float: right;
    }

```

```
.loading {  
    background: url("../img/loadingAnimation.gif") no-repeat scroll center  
                center white;  
    display: none;  
    height: 100%;  
    opacity: 0.7;  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 100%;  
}
```