ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

თორნიკე აბრამიშვილი

კრიპტოგრაფიული მეთოდების იმპლემენტაცია ელექტრონულ კანცელარიაში

კომპიუტერული მეცნიერება

ნაშრომი შესრულებულია კომპიუტერული მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: მიხეილ თუთბერიძე, ფიზ.-მათ. მეცნ. კანდიდატი

თბილისი

2018

სარჩევი

ანოტაცია

წინამდებარე ნაშრომის ფარგლებში შემუშავებულია ელექტრონული კანცელარიის ფუნქციონალის მქონე პროგრამული უზრუნველყოფა, რომელიც დოკუმენტზე ხელმოწერის ავთენტურობის დასადასტურებლად იყენებს კრიპტოგრაფიულ მეთოდებს. სისტემა წარმოადგენს ვებ აპლიკაციას და შესაძლებელია მისი გამოყენება სხვადასხვა მასშტაბის ორგანიზაციების საკანცელარიო საქმიანობაში.

Abstract

Tornike Abramishvili

Implementing Cryptographic Methods in Electronic Chancellery

In the scope of the present work the software with electronic chancellery functional is developed which uses cryptographic methods for confirming authentity of documents signature. System is created as web application and it is possible to use this system in chancelleries of organizations with various scale.

შესავალი

ბიზნესის და ინფორმაციული ტექნოლოგიების თანამშრომლობა უფრო და უფრო ეფექტური ხდება და მნიშვნელოვან ბიძგს აძლევს ინფორმაციული ტექნოლოგიების ახალ-ახალი მიმართულებებით განვითარებას. დღევანდელი ბიზნესის წარმადობა მნიშვნელოვნად არის დამოკიდებული ინფორმაციულ ტექნოლოგიებზე და საკმაოდ დიდი თანხები იხარჯება მათ დანერგვაზე.

ორგანიზაციების უმრავლესობა მნიშვნელოვნად დამოკიდებულია ინფორმაციული ტექნოლოგიების გამართულად მუშაობაზე. ინფორმაციული ტექნოლოგიები არსებითად ასწრაფებს ბიზნესპროცესებს, ორგანიზაციას თავიდან აცილებს რუტინულ შრომას, დააზოგინებს შრომით რესურსებს. მაგრამ ასევე მნიშვნელოვანია, რომ დაზოგილ იქნას მატერიალური რესურსები. ამ თვალსაზრისით კი განსაკუთრებული აღნიშვნის ღირსია ორგანიზაციის შიდა დოკუმენტბრუნვის პროცესი, რომლის დროსაც უამრავი ქაღალდი და სხვა საკანცელარიო მასალა იხარჯება.

ორგანიზაციაში დოკუმენტბრუნვას უზრუნველყოფს საკანცელარიო სამსახური. კანცელარიის ამოცანებში შედის შემომავალი და გამავალი დოკუმენტაციის აღრიცხვა, ასევე ორგანიზაციის შიდა სტრუქტურული ერთეულებს შორის დოკუმენტების მოძრაობის აღრიცხვა. აღნიშნული ამოცანების სპეციალიზებული კომპიუტერული პროგრამის გარეშე გადაჭრისას აუცილებელია საკმაოდ დიდი ადამიანური რესურსი და ასევე საკმაოდ დიდი მოცულობით ხარჯვადი მატერიალური რესურსი.

ორგანიზაციის დოკუმენტბრუნვისას ადამიანური და მატერიალური რესურსების ყაირათიანად ხარჯვის ამოცანის გადასაწყვეტად შესაძლებელია გამოყენებულ იქნას ელექტრონული კანცელარიის ფუნქციონალის მქონე პროგრამული უზრუნველყოფა. ამგვარი პროგრამული უზრუნველყოფა ავტომატურ რეჟიმში განახორციელებდა ბევრ ისეთ ოპერაციას, რასაც კანცელარიაში ადამიანები აკეთებენ (მაგალითად - შემომავალი და გამავალი დოკუმენტაციის რეგისტრაცია) და ასევე, ინფორმაციას შეინახავდა მყარ დისკზე, ნაცვლად ქაღალდებისა. თუმცა, ამ შემთხვევაში გადასაწყვეტი ხდება დოკუმენტის ხელმოწერის ავთენტურობის დადასტურების

პრობლემა, რაშიც შეიძლება გამოყენებულ იქნას ხელმოწერის კრიპტოგრაფიული მეთოდი - ციფრული ხელმოწერა.

გამოქვეყნებულია მრავალი ნაშრომი, რომელშიც ციფრული ხელმოწერა არის განხილული.

[1] ნაშრომში შემოთავაზებულია მიდგომა, რომელიც იძლევა იმის გარანტიას, რომ შეტყობინების გადაცემისას მისი ავტორის იდენტურობა არ შეცვლილა ხელმოწერის შემდგომ.

[2] ნაშრომში განხილულია ორ პირს შორის არსებული მიმოწერისას თითოეული მათგანის მესამე პირის მიერ ჩანაცვლების პრობლემა და შემოთავაზებულია ციფრული ხელმოწერის გამოყენება ამ მიზნით.

[3] ნაშრომში შემოთავაზებულია ავთენტიკაციის მექანიზმი ციფრული ხელმოწერის გამოყენებით.

გარდა ამისა, აღნიშნულ თემას ეძღვნება ნაშრომები: [4], [5], [67], [77777] და სხვა მრავალი.

წინამდებარე ნაშრომზე მუშაობის ფარგლებში შემუშავებულია ელექტრონული კანცელარიის ფუნქციონალის მქონე პროგრამული უზრუნველყოფა, რომელშიც დოკუმენტებზე ხელმოწერის მიზნით გამოყენებულია ციფრული ხელმოწერის მექანიზმი.

ამოცანის დასმა

როგორც უკვე შესავალში იქნა აღნიშნული, წინამდებარე ნაშრომის მიზანს წარმოადგენს ელექტრონული კანცელარიის ფუნქციონალის მქონე პროგრამული უზრუნველყოფის შემუშავება, რომელშიც დოკუმენტებზე ხელმოწერის მიზნით გამოყენებულია ციფრული ხელმოწერის მექანიზმი. აპლიკაცია უნდა შეიქმნას, როგორც ვებ აპლიკაცია, რომელიც თავის მონაცემებს განათავსებს რელაციურ მონაცემთა ბაზაში. გარდა ამისა, ელექტრონული კანცელარიას შეიძლება გააჩნდეს შემდეგი ფუნქციები:

1. ორგანიზაციის საქმიანობისას გამოცემული შიდა აქტების აღრიცხვა, გადამისამართება შესაბამის თანამშრომელზე და აქტის შინაარსიდან გამომდინარე, მისთვის საბოლოო სტატუსის მინიჭებამდე კონტროლის დაწესება.

2. შემოსული კორესპონდენციის აღრიცხვა და შესაბამის თანამდებობის პირზე გადამისამართება, რომელიც მასზე მოახდენს რეაგირებას, ან გადაამისამართებს სხვა თანამდებობის პირთან. შემოსულ კორესპონდენციაზე ასევე აუცილებელია კონტროლის დაწესება, სანამ მას საბოლოო სტატუსი მიენიჭება.

ზემოთაღნიშნული ოპერაციების განსახორციელებლად თანამდებობის პირები დოკუმენტზე რეაგირებისას მას ადებენ რეზოლუციას და აწერენ ხელს. რეზოლუციის დადება და ხელმოწერა ხორციელდება ციფრულად, კრიპტოგრაფიული მეთოდების გამოყენებით.

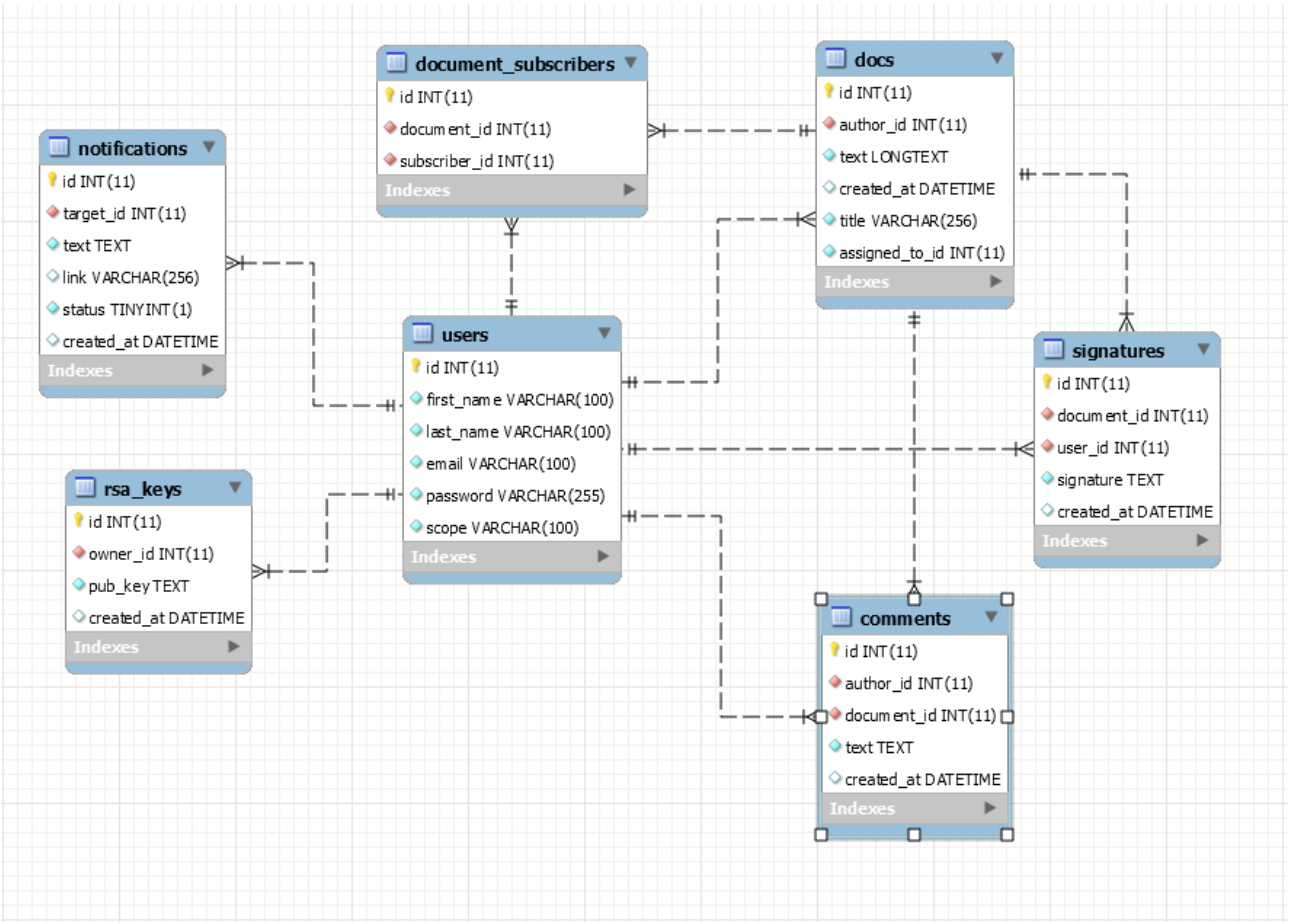
ცნობილია, რომ ციფრული ხელმოწერისთვის არსებობს ორნაირი გასაღები: ღია და დახურული. ღია გასაღები, ცხადია, რომ უნდა ყველასთვის იყოს ხელმისაწვდომი და იგი უნდა შენახულ იქნას ელექტრონული კანცელარიის მონაცემთა ბაზაში. რაც შეეხება დახურულ გასაღებს, მისი შენახვა მონაცემთა ბაზის მხარეს არ არის უსაფრთხო, რადგან მასზე წვდომა ექნება მონაცემთა ბაზის მომსახურე ტექნიკურ პერსონალს. ამიტომ აუცილებელია, რომ დახურული გასაღები ინახებოდეს მომხმარებლის მხარეს და შესაბამისად, ტექსტის ციფრული ხელმოწერა უნდა ხდებოდეს მომხმარებლის მხარეს, კერძოდ, მომხმარებლის ბრაუზერში.

თანამდებობის პირი, მიიღებს რა დოკუმენტს რეზოლუციის დასადებად, შეიმუშავებს რეზოლუციის ტექსტს და შემდეგ მიმართავს თავის კომპიუტერში შენახულ დახურულ გასაღებს, რომლის მეშვეობითაც განახორციელებს ხელმოწერას, ხოლო შემდეგ ამ ხელმოწერას ატვირთავს სისტემაში.

გამოყენებული ტექნოლოგიები

პროგრამული უზრუნველყოფა შესრულდა NodeJS პლატფორმის გამოყენებით, ხოლო ინფორმაციის სტრუქტურიზებულად შესანახად გამოყენებულია რელაციური მონაცემთა ბაზა, კერძოდ - MySQL-ის მონაცემთა ბაზა. კლიენტის მხარეს გამოყენებულია VueJs ფრეიმვორქი და მისი საშუალებით არის შექმნილი SPA (Single Page Application). ასევე გამოყენებულია Nuxt ფრეიმვორქი რომელიც ახდენს SSR (Server-Side-Rendering)-ს, მომხმარებლის მიერ მოთხოვნილი გვერდის სერვერის მხარეს სრულად დაგენერირებას და მისთვის სრული გვერდის მიწოდებას.

მონაცემთა ბაზის სტრუქტურა



ცხრილები

```sql
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(100) NOT NULL,
  `last_name` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  `scope` varchar(100) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email_UNIQUE` (`email`),
  KEY `user_email_ind` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8;
```

დარეგისტრირებული მომხმარებლების ცხრილი. ინახება მომხმარებლის ინფორმაცია. ინდექსი ადევს email-ს რათა უკეთ მოხდეს email-ის მიხედვით მოძებნა.

```
CREATE TABLE `docs` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `author_id` int(11) NOT NULL,
  `text` longtext NOT NULL,
  `created_at` datetime DEFAULT NULL,
  `title` varchar(256) NOT NULL,
  `assigned_to_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `docs_author_id_ind` (`author_id`),
  KEY `docs_ibfk_1_idx` (`assigned_to_id`),
  CONSTRAINT `docs_ibfk_1` FOREIGN KEY (`author_id`) REFERENCES `users`
(`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8;
```

დოკუმენტების ცხრილი. ინახება ინფორმაცია დოკუმენტის შესახებ და აქვს გარე გასაღები მომხმარებლების ცხრილზე რომელიც მიუთითებს დოკუმენტის ავტორს.

```
CREATE TABLE `rsa_keys` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `owner_id` int(11) NOT NULL,
  `pub_key` text NOT NULL,
  `created_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `keys_owner_id_ind` (`owner_id`),
  CONSTRAINT `rsa_keys_ibfk_1` FOREIGN KEY (`owner_id`) REFERENCES `users`
(`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
```

RSA გასაღებების ცხრილი. ინახება მომხმარებლების მიერ დაგენერირებული საჯარო გასაღებები (public key). აქვს გარე გასაღები მომხმარებლების ცხრილზე რომელიც მიუთითებს ამ გასაღების მფლობელს.

```
CREATE TABLE `signatures` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `document_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `signature` text NOT NULL,
  `created_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `document_id` (`document_id`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `signatures_ibfk_1` FOREIGN KEY (`document_id`) REFERENCES
`docs` (`id`) ON DELETE CASCADE,
  CONSTRAINT `signatures_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users`
(`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8;
```

ხელმოწერების ცხრილი ინახავს ინფორმაციას დოკუმენტზე გაკეთებული ხელმოწერის შესახებ. აქვს გარე გასაღები:

1. დოკუმენტების ცხრილზე, რომელიც მიუთითებს რომელ დოკუმენტზე მოხდა ხელმოწერა.
2. მომხმარებლების ცხრილზე, რომელიც მიუთითებს რომელმა მომხმარებელმა მოაწერა ხელი.

```
CREATE TABLE `comments` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `author_id` int(11) NOT NULL,
  `document_id` int(11) NOT NULL,
  `text` text NOT NULL,
  `created_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `comments_document_id_ind` (`document_id`),
  KEY `author_id` (`author_id`),
  CONSTRAINT `comments_ibfk_1` FOREIGN KEY (`document_id`) REFERENCES `docs`
(`id`) ON DELETE CASCADE,
```

```
  CONSTRAINT `comments_ibfk_2` FOREIGN KEY (`author_id`) REFERENCES `users`
(`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=37 DEFAULT CHARSET=utf8;
```

კომენტარების ცხრილი ინახავს ინფორმაციას დოკუმენტზე გაკეთებული კომენტარების შესახებ. აქვს გარე გასაღები:

1. მომხმარებლების ცხრილზე, რომელიც მიუთითებს რომელმა მომხმარებელმა გააკეთა კომენტარი.
2. დოკუმენტების ცხრილზე, რომელიც მიუთითებს რომელ დოკუმენტზე გაკეთდა კომენტარი.

```
CREATE TABLE `document_subscribers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `document_id` int(11) NOT NULL,
  `subscriber_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `doc_subscribers_ibfk_1_idx` (`subscriber_id`),
  KEY `doc_subscribers_ibfk_2_idx` (`document_id`),
  CONSTRAINT `doc_subscribers_ibfk_1` FOREIGN KEY (`subscriber_id`)
REFERENCES `users` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `doc_subscribers_ibfk_2` FOREIGN KEY (`document_id`) REFERENCES
`docs` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;
```

დოკუმენტებზე გამოწერის ცხრილი. ინახავს ინფორმაციას თუ რომელი დოკუმენტებისთვის რომელმა მომხმარებლებმა უნდა მიიღონ სიახლეების შეტყობინება. აქვს გარე გასაღები:

1. მომხმარებლების ცხრილზე, რომელიც მიუთითებს რომელმა მომხმარებელმა უნდა მიიღოს ინფორმაცია.
2. დოკუმენტების ცხრილზე, რომელიც ინახავს ინფორმაციას რომელი დოკუმენტის სიახლეზე უნდა მოხდეს მომხმარებლისთვის შეტყობინების გაგზავნა.

```
CREATE TABLE `notifications` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `target_id` int(11) NOT NULL,
  `text` text NOT NULL,
  `link` varchar(256) DEFAULT NULL,
  `status` tinyint(1) NOT NULL DEFAULT '0',
  `created_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `notifications_target_id_ind` (`target_id`),
  CONSTRAINT `notifications_ibfk_1` FOREIGN KEY (`target_id`) REFERENCES
`users` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=49 DEFAULT CHARSET=utf8;
```

შეტყობინებების ცხრილი ინახავს ინფორმაციას მომხმარებლებისთვის საინტერესო სიახლეების შესახებ. აქვს გარე გასაღები მომხმარებლების ცხრილზე, რომელიც ინახავს ინფორმაციას თუ რომელ მომხმარებელს უნდა მიუვიდეს ეს შეტყობინება.

პროცედურები

```
DELIMITER $$
CREATE PROCEDURE `create_notifications_for_signature`(in document_id int, in
author_id int)
BEGIN

    DECLARE author_name varchar(256);
    DECLARE document_title varchar(256);
    DECLARE sub_id int;

    DECLARE tCount INT;

    DECLARE bDone INT;
    DECLARE curs CURSOR FOR SELECT ds.subscriber_id FROM
Document_Subscribers ds WHERE ds.document_id = document_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET bDone = 1;
```

```
    DROP TEMPORARY TABLE IF EXISTS tblResults;
    CREATE TEMPORARY TABLE IF NOT EXISTS tblResults  (
        auth_id int
    );

    select concat(first_name, " ", last_name) into author_name from Users
where id=author_id;
    select title into document_title from Docs where id=document_id;

    OPEN curs;

    SET bDone = 0;
    REPEAT
    FETCH curs INTO sub_id;

    IF sub_id <> author_id then
        select count(*) into tCount from tblResults where auth_id=sub_id;
        if tCount = 0 then
            INSERT INTO Notifications (target_id, `text`, link, created_at)
VALUES (sub_id, concat(author_name, ' has signed on ', document_title),
concat('/documents/', document_id), now());
            INSERT INTO tblResults values (sub_id);
        END if;
    END IF;
    UNTIL bDone END REPEAT;

    CLOSE curs;
    DROP TEMPORARY TABLE IF EXISTS tblResults;

END$$
DELIMITER ;
```

ეს პროცედურა პასუხისმგებელია რომ კონკრეტული დოკუმენტის გამომწერებისთვის შექმნას ახალი შეტყობინებები დოკუმენტზე ხელმოწერის გაკეთების შესახებ.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`create_notifications_for_new_comment`(in document_id int, in author_id int)
BEGIN

    DECLARE author_name varchar(256);
    DECLARE document_title varchar(256);
    DECLARE sub_id int;

    DECLARE tCount INT;

    DECLARE bDone INT;
    DECLARE curs CURSOR FOR SELECT ds.subscriber_id FROM
Document_Subscribers ds WHERE ds.document_id = document_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET bDone = 1;

    DROP TEMPORARY TABLE IF EXISTS tblResults;
    CREATE TEMPORARY TABLE IF NOT EXISTS tblResults  (
        auth_id int
    );

    select concat(first_name, " ", last_name) into author_name from Users
where id=author_id;
    select title into document_title from Docs where id=document_id;

    OPEN curs;

    SET bDone = 0;
    REPEAT
    FETCH curs INTO sub_id;

    IF sub_id <> author_id then
        select count(*) into tCount from tblResults where auth_id=sub_id;
        if tCount = 0 then
```

```
            INSERT INTO Notifications (target_id, `text`, link, created_at)
VALUES (sub_id, concat(author_name, ' has written a new comment on ',
document_title), concat('/documents/', document_id), now());
            INSERT INTO tblResults values (sub_id);
        END if;
    END IF;
    UNTIL bDone END REPEAT;

    CLOSE curs;
    DROP TEMPORARY TABLE IF EXISTS tblResults;

END$$
DELIMITER ;
```

ეს პროცედურა პასუხისმგებელია რომ კონკრეტული დოკუმენტის გამომწერებისთვის შექმნას ახალი შეტყობინებები დოკუმენტზე გაკეთებული კომენტარის შესახებ.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`create_notifications_for_document_assignment`(in document_id int, in
assignee_id int)
BEGIN

    DECLARE assignee_name varchar(256);
    DECLARE document_title varchar(256);
    DECLARE sub_id int;

    DECLARE tCount INT;

    DECLARE bDone INT;
    DECLARE curs CURSOR FOR SELECT ds.subscriber_id FROM
Document_Subscribers ds WHERE ds.document_id = document_id;
```

```
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET bDone = 1;

    DROP TEMPORARY TABLE IF EXISTS tblResults;
    CREATE TEMPORARY TABLE IF NOT EXISTS tblResults  (
        auth_id int
    );

    select concat(first_name, " ", last_name) into assignee_name from Users
where id=assignee_id;
    select title into document_title from Docs where id=document_id;

    OPEN curs;

    SET bDone = 0;
    REPEAT
    FETCH curs INTO sub_id;

    IF sub_id <> assignee_id then
        select count(*) into tCount from tblResults where auth_id=sub_id;
        if tCount = 0 then
            INSERT INTO Notifications (target_id, `text`, link, created_at)
VALUES (sub_id, concat(document_title, ' was assigned to ', assignee_name),
concat('/documents/', document_id), now());
            INSERT INTO tblResults values (sub_id);
        END if;
    END IF;
    UNTIL bDone END REPEAT;
    INSERT INTO Notifications (target_id, `text`, link, created_at) VALUES
(assignee_id, concat(document_title, ' was assigned to YOU'),
concat('/documents/', document_id), now());

    CLOSE curs;
    DROP TEMPORARY TABLE IF EXISTS tblResults;

END$$
```

```
DELIMITER ;
```

ეს პროცედურა პასუხისმგებელია რომ კონკრეტული დოკუმენტის გამომწერებისთვის და კონკრეტული ოპერატორისთვის შექმნას ახალი შეტყობინებები დოკუმენტის ამ ოპერატორზე მინიჭებისას.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_subscriber`(in document_id
int, in author_id int)
BEGIN

    DECLARE tCount int;

    select count(*) into tCount from Document_Subscribers ds where
ds.document_id=document_id and ds.subscriber_id=author_id;
    if tCount = 0 then
        INSERT INTO Document_Subscribers (subscriber_id, document_id) values
(author_id, document_id);
    end if;

END$$
DELIMITER ;
```

ეს პროცედურა პასუხისმგებელია რომ გააკეთოს უნიკალური ჩანაწერი დოკუმენტის გამოწერაზე.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `change_assignee`(in document_id
int, in assignee_id int)
BEGIN

    UPDATE Docs SET assigned_to_id=assignee_id where id=document_id;
```

```
    call create_notifications_for_document_assignment(document_id,
assignee_id);

END$$
DELIMITER ;
```

ეს პროცედურა პასუხისმგებელია რომ მოახდინოს დოკუმენტის მინიჭება ოპერატორზე და შესაბამისი შეტყობინებების დაგენერირება.

ტრიგერები

```
CREATE DEFINER=`root`@`localhost` TRIGGER after_comments_insert
AFTER INSERT ON Comments
FOR EACH ROW BEGIN

    DECLARE tCount int;

    call create_notifications_for_new_comment(NEW.document_id,
NEW.author_id);
    call add_subscriber(NEW.document_id, NEW.author_id);
END
```

დოკუმენტზე კომენტარის გაკეთების შემდეგ ეს ტრიგერი ქმნის ახალ შეტყობინებებს ამ დოკუმენტის გამომწერებისთვის და კომენტარის გამკეთებელ მომხმარებელს ამატებს დოკუმენტის გამომწერების სიაში თუკი უკვე არ აქვს გამოწერილი.

```
CREATE DEFINER=`root`@`localhost` TRIGGER after_documents_insert
AFTER INSERT ON Docs
FOR EACH ROW BEGIN

```

```
    INSERT INTO Document_Subscribers (subscriber_id, document_id) values
(NEW.author_id, NEW.id);

END
```

დოკუმენტის შექმნის შემდეგ ეს ტრიგერი დოკუმენტის შემქმნელს ამატებს ამ დოკუმენტების გამოწერის სიაში.

```
CREATE DEFINER=`root`@`localhost` TRIGGER after_signature_insert
AFTER INSERT ON Signatures
FOR EACH ROW BEGIN

    call create_notifications_for_signature(NEW.document_id, NEW.user_id);
    call add_subscriber(NEW.document_id, NEW.user_id);

END
```

დოკუმენტზე ხელმოწერის გაკეთების შემდეგ ეს ტრიგერი ქმნის ახალ შეტყობინებებს ამ დოკუმენტის გამომწერებისთვის და ხელმოწერის გამკეთებელ მომხმარებელს ამატებს დოკუმენტის გამომწერების სიაში თუკი უკვე არ აქვს გამოწერილი.

პროგრამული კოდის საკვანძო ფრაგმენტები და აღწერა

public/private key-ების დაგენერირება:

```
generateKeyPair: function(algorithm, length) {
        const pair = KEYUTIL.generateKeypair(algorithm, length);
        return {
            prvKey: KEYUTIL.getPEM(pair.prvKeyObj, "PKCS1PRV"),
            pubKey: KEYUTIL.getPEM(pair.pubKeyObj)
        };
    }
```

private key-ს გამოყენებით ხელმოწერის დაგენერირება:

```
signData: function(algorithm, prvKey, data){
        var sig = new KJUR.crypto.Signature({
            alg: algorithm
        });
        sig.init(prvKey);
        sig.updateString(data.toString());
        return sig.sign();
    }
```

public key-ის გამოყენებით მოცემული ხელმოწერის ვალიდურობის დადგენა:

```
verifyData: function(algorithm, pubKey, data, signature){
        var sig = new KJUR.crypto.Signature({
            alg: algorithm
        });
        sig.init(pubKey);
        sig.updateString(data.toString());
        return sig.verify(signature);
    }
```

ზემოთ მოყვანილი კოდის ფრაგმენტები არის გაშვებული WebWorker-ში რადგან გასაღებების დაგენერირებისას ას მოხდეს ჯავასკრიპტის მთავარი ნაკადის დაბლოკვა. შიფრაციის გაკეთებისას გამოიყენება RSA ალგორითმი, ხოლო ხელმოწერისას SHA512.

დასკვნა

ამრიგად, წინამდებარე ნაშრომის ფარგლებში შექმნილია ვებ აპლიკაცია, რომელსაც გააჩნია ელექტრონული კანცელარიის ფუნქციონალი და რომელიც დოკუმენტზე ხელმოსაწერად იყენებს ციფრული ხელმოწერის მექანიზმს. სისტემაში უზრუნველყოფილია ციფრული ხელმოწერის მაქსიმალური სანდოობა და მინიმუმამდეა დაყვანილი ხელმოწერის გაყალბების შესაძლებლობა ტექნიკური პერსონალის ან ბოროტმოქმედის მიერ. სისტემა მზად არის ექსპლოატაციაში გასაშვებად.

გამოყენებული ლიტერატურა

1. Obaidat, M., & Boudriga, N. (2007). Authentication and digital signature. In Security of e-Systems and Computer Networks (pp. 48-72). Cambridge: Cambridge University Press.

2. Talbot, J., & Welsh, D. (2006). Digital signatures. In Complexity and Cryptography: An Introduction (pp. 170-186). Cambridge: Cambridge University Press.

3. Mason, S. (2012). Digital signatures. In Electronic Signatures in Law (Law Practitioner Series, pp. 259-302). Cambridge: Cambridge University Press.

4. Goldreich, O. (2004). Digital Signatures and Message Authentication. In Foundations of Cryptography (pp. 497-598). Cambridge: Cambridge University Press.

5. Germany: Digital Signatures Ordinance. (1998). International Legal Materials, 37(3), 579-586.

6. Galbraith, S. (2012). Digital signatures based on discrete logarithms. In Mathematics of Public Key Cryptography (pp. 452-468). Cambridge: Cambridge University Press.

7. Klein, P. (2014). Public-Key Cryptosystems and Digital Signatures. In A Cryptography Primer: Secrets and Promises (pp. 157-170). Cambridge: Cambridge University Press.