



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

# **SITE**

## **ONLINE VOTING SYSTEM**

DATABASE MANAGEMENT SYSTEM

ITE1003

UNDER: PROF. GOVINDA K

**18BIT0195**

RISHI RAJ GUPTA



## **ABSTRACT**

The voting system is the backbone of every democracy and organization. The main goal of voting (in a scenario involving the citizens of a given country) is to come up with leaders of the people's choice. There are various voting techniques used such as Paper Ballot Voting System, E-Voting System also known as Electronic Voting System, Internet Voting System, SMS and Miss Calls Voting System.

Some of the problems involved include rigging votes during election, insecure or inaccessible polling stations, inadequate polling materials and also inexperienced personnel. In this paper, we have tried to sought out issues faced by multi various voting systems and make it work multiplatform on any operating system.


This online voting/polling system seeks to address the above issues. It should be noted that with this system in place, the users, citizens in this case shall be given ample time during the voting period. They shall also be trained on how to vote online before the election time.



## INTRODUCTION

“ONLINE VOTING SYSTEM” is an online voting technique. In this system people who have citizenship of India and whose age is above 18 years of age and any sex can give his\her vote online without going to any physical polling station. There is a database which is maintained in which all the names of voters with complete information is stored.

In “ONLINE VOTING SYSTEM” a voter can use his\her voting right online without any difficulty. He\She has to be registered first for him/her to vote. Authorization is mainly done by the system administrator for security reasons. The system Administrator registers the voters on a special site of the system visited by him only by simply altering the username of the valid registered user by a system generated unique id . Citizens seeking registration are expected to contact the system administrator to submit their details. After the validity of them being citizens of India has been confirmed by the system administrator by comparing their details submitted with those in existing databases such as those as the Registrar of Persons, the citizen is then registered as a voter.



After registration, the voter is assigned an encrypted username with which he/she can use to log into the system and enjoy services provided by the system such as voting. If invalid/wrong details are submitted, then the citizen is not registered to vote.


## MOTIVATION

The voting system is a process of selecting the right candidate for the development of any government and organization. This system must ensure integrity and secrecy of the votes been cast (without casting multiple votes). The motivation behind this project includes-

- The main goal of the idea proposed is to encourage more people to vote remotely where still the voting percentage lags behind 60 percentage of the total population
- Ending the current process of queue system and making the voting process efficient and available to larger section of the population
- This idea is geared towards more secure and privileged level voting where every vote counts the worth of the voter and the candidate
- Reduction in the cost of the manual voting system will save the country's money

## ISSUES IN THE EXISTING SYSTEM

- **Expensive and Time consuming:** The process of collecting data and entering this data into the database takes too much time and is expensive to conduct
- **Loss of registration forms:** Sometimes, registration forms get lost after being filled in with voters' details, in most cases these are difficult to follow-up and therefore many remain unregistered even though they are voting age nationals and interested in exercising their right to vote.
- **Complexity in the proposed voting system:** There is a need to develop a idea for online voting system which is simple and understood by all the sections of the society
- **Security problems:** Foreign experience revealed that they are often confronted by security issues while the online voting system is running. The origin of the security issues was due to not only outsider (such as voters and attackers) but also insider (such as system developers and



administrators), even just because the inheritance of some objects in the source code are unsuitable. These errors caused the voting system to crash.

## PROPOSED PROJECT-Overview

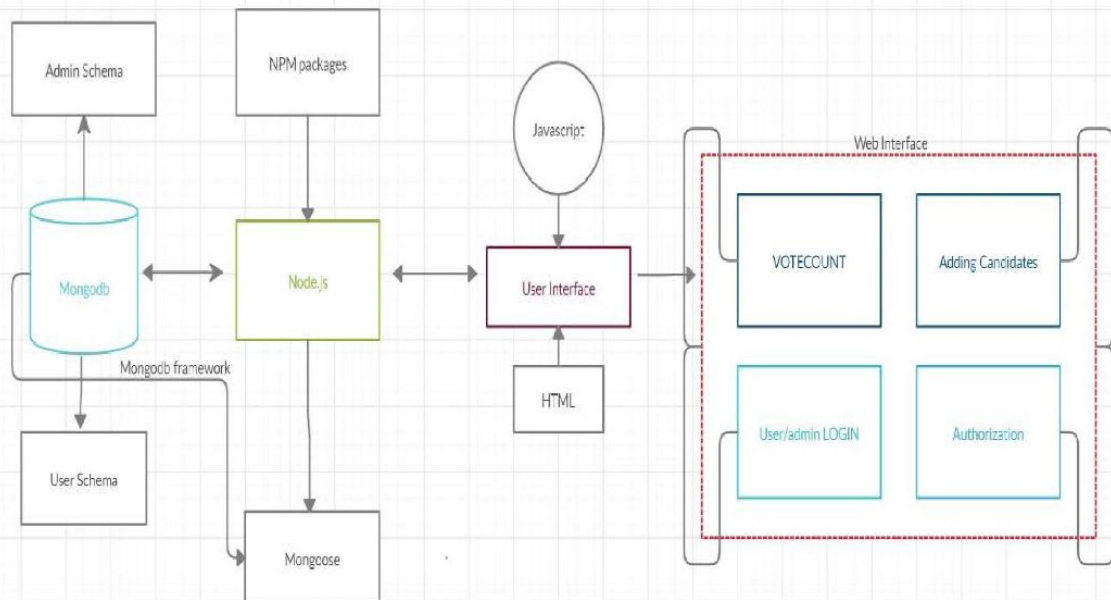
How people can find a reliable interface for casting votes?

- The proposed idea consists that the election commission will have a web interface through which people should register using their unique identification number i.e. Adhaar id and validate their process to cast vote.
- A voter id will be provided to every people as an encrypted id with the username which will be unique for every registered voter
- For casting votes they should login with their unique user id and password and select their preferable party member .The casted vote will be saved in the administrator database for further counting
- After casting vote if a user wants to further access the voting process, it will be denied.

## LITERATURE SURVEY

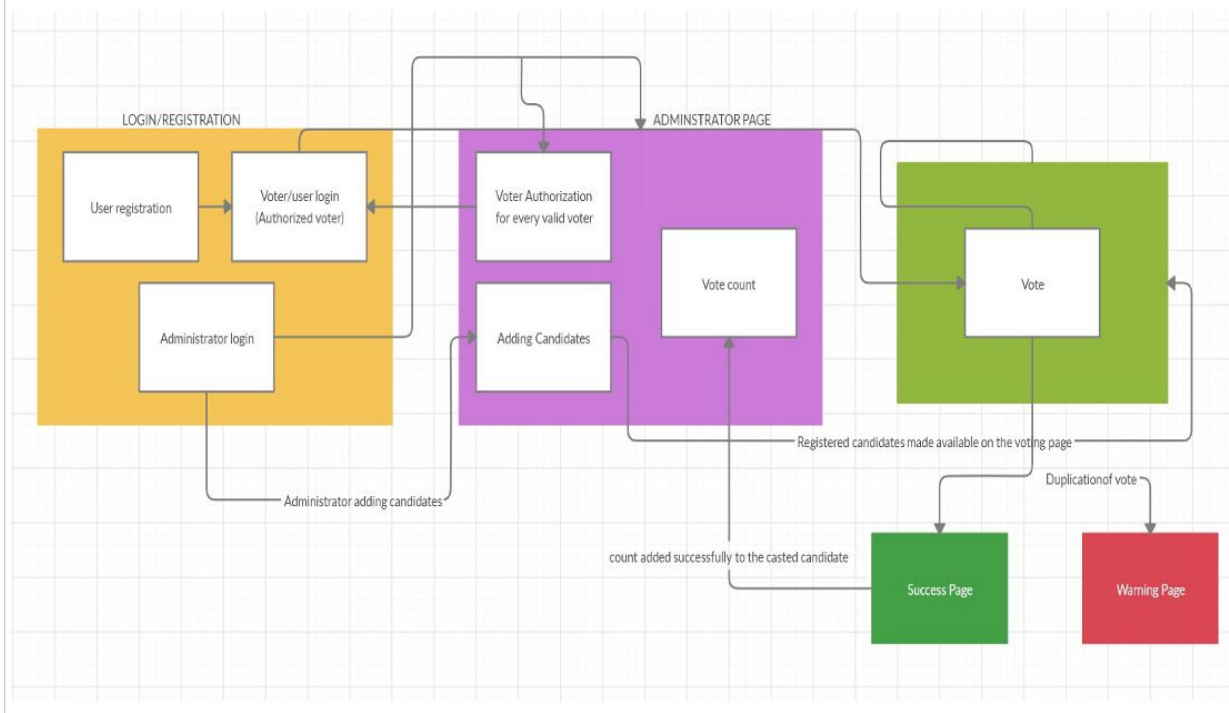
Authors and Year	Title	Conceptual Model
<p>Prof. Anisaara Nadaph, Ashmita Katiyar, Tushar Naidu, Rakhi Bondre, Durgesh Kumari Goswami</p> <p>YEAR AND JOURNAL- International Journal of  Innovative Research in Computer Science &amp; Technology (IJIRCST), September 2015</p>	<p><b>AN ANALYSIS OF SECURE ONLINE VOTING SYSTEM</b></p> <p><b>MULTI-PURPOSE PLATFORM INDEPENDENT ONLINE VOTING SYSTEM</b></p>	<p>In this system, voter can cast the vote by sending SMS to the system using any kind of hand set through the mobile switching center.</p> <p>The voter has to first dial a toll free number in which he has to answer through SMS depending upon the questions asked on recorded call.</p>

## SYSTEM ARCHITECTURE





## FUNCTIONAL ARCHITECTURE



## MODULE WISE DESCRIPTION

### MODULE-1 DATABASE DESIGN

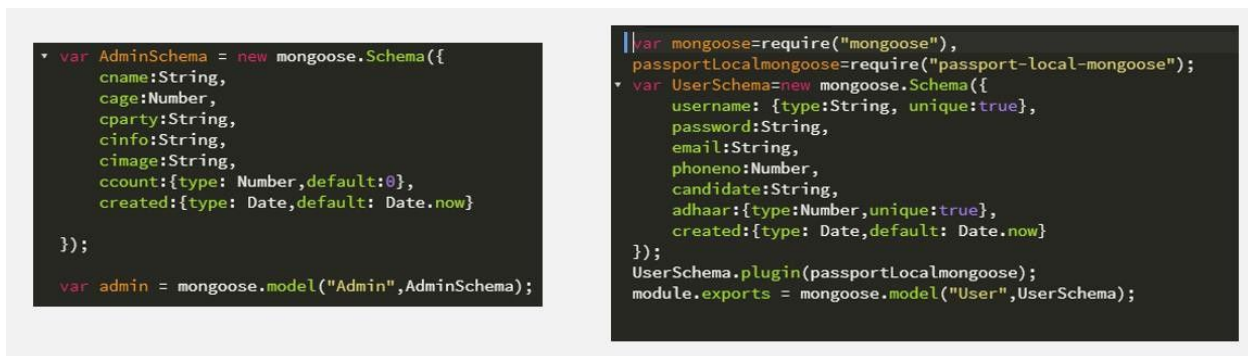


Figure-i

The module describes the database structure and design which uses two structured schemas or here collections- admin schema and user schema.

Admin schema describes the document related to candidate details and vote count for that candidate handled by the administrator. User schema describes the document related to the user/voter details which is incorporated by the registering through a register form and validating every user by the administrator

## MODULE-2 REGISTRATION/LOGIN

```
app.post("/register",function(req,res){
  User.register(
    new User({
      username:req.body.username,
      email:req.body.email,
      phoneno:req.body.phoneno,
      adhaar:req.body.adhaar,
    }),
    req.body.password, function(err,user){
      if(err){
        console.log(err);
        return res.render("register");
      }
      passport.authenticate("local")(req,res,function(){
        res.redirect("/");
      });
    });
});
```

```
//ADMIN LOGIN
app.get("/login/admin",function(req,res){
  res.render("adminlogin");
});

app.post("/login/admin",function(req,res){
  if(req.body.username=="shaswat" && req.body.password=="123"){
    res.redirect("/admin");
  }else{
    res.redirect("/login/admin");
  }
});
```

Figure-j

The module describes the backend mechanism of registration and login in node.js using npm package passport.js. The user registers itself filling in the details provided in the form and logs in with the username provided by the administrator.

Administrator login is defined explicitly as there exist a single handler to administrator page

### MODULE-3 AUTHORIZATION

```
<div class="field">
  <label>Username</label>
  <input type="text" name="voter[username]" value="<%=voter.username%><%=voter._id%>">
</div>
```

Figure-k

As depicted in figure-4 and 5, the authorization is handled by the administrator encrypting a unique system generated voter id to the username of the voter

## MODULE-4 VOTING/VOTECOUNT

This module describes the voting logic and the counting of the votes for every candidate standing in the election.

```
User.findById(req.user._id,function(err,user){
  if(err){
    console.log(err);
  }else{
    admin.findById(req.params.id,function(err,foundcandidate){
      if(err){
        res.render("votepage");
      }else {
        if(user.candidate!=undefined){
          res.render("warningage");
        }else{
          console.log("posted!!");
          console.log(user);
          console.log(foundcandidate);
          console.log("done!");
          user.candidate=foundcandidate;
          user.save();
          foundcandidate.ccount+=1;
          foundcandidate.save(function(err){
            if(err){
              console.log(err);
            }
          })
          res.render("successage",{candidate:foundcandidate});
        }
      }
    }
  }
})
```

Figure-1

The authorized user will be allowed to the voting portal as shown in figure-6. The voting option for candidates will redirect through the success page if the voter has voted once. Else, he will be redirected to the warning page which shows the rejection of the casted vote.

Voter's privacy is maintained in the database as the voted

## INNOVATIVE IDEA/NOVELTY

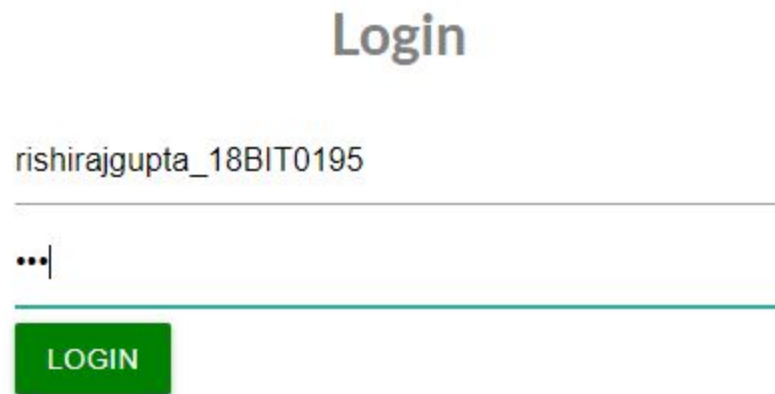
The proposed idea for voting provides

- Simple voting technique which will enable , motivate larger sections to cast for votes
- Secure and trustworthy as it works on a single admin source through which people are directly related
- Prohibition of interference of third party to jeopardise the voting system
- The idea uses mongodb instead of mysql as-
  1. MongoDB uses a role-based access control with a flexible set of privileges. Its security features include authentication, auditing, and authorization
  2. It supports high availability of data with automatic, fast an instant data recovery
  3. In MongoDB, documents are able to have their own unique structure. New fields can be added at any time and contain any type of value.

## IMPLEMENTATION DETAILS

### SCREENSHOTS WITH DETAILED EXPLANATION

---



Initially, the admin will login through his specified username and password as depicted which will render him through to the administrator page which follows privileged actions

After logging in the admin will have to add candidates details through a form provided in the administrator page

### New Candidate

Candidate Name  
RISHI RAJ GUPTA

Candidate Age  
21

Party name  
NDA

Vote count  
0

Candidate Information  
aaaaaaaaaaaaasssssssssbbbbb\n  
dsdjs  
dsdjsjd

Candidate Photo  
Candidate photo

Submit

The candidate details will be saved in the database through which the voter will be able to view the details in the vote page and cast their vote after registration/authorization process. A vote count attribute is defined for every candidate which is fixed to 0 initially..



## Sign up

RISHI RAJ

.....

rishiraj.gupta2018@vitstudent.ac.in

45500000000

1234567890|

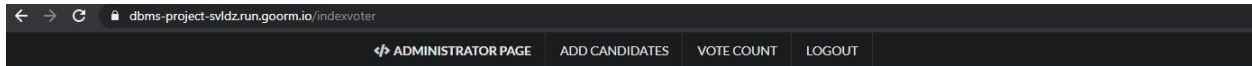
CREATE

Already an user? [Login Here](#)

After the admin has added the details of the candidate the user can now register which after registration will be qualified as registered voter. (Voting is only allowed for authorized users from admin)

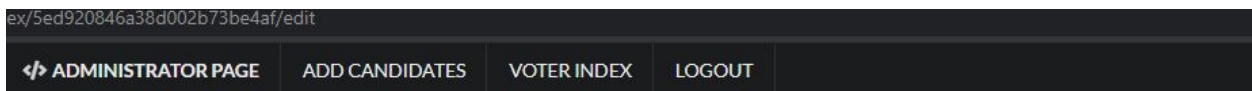
After the user has registered, admin now has the option to authorize or remove the user based on the adhaar number on the records and the user creation timing as depicted .

The details of the voter in the form of adhaar number will already be registered in the administrator system. Admin can verify from the available documents of the citizens/voters and authorize the same.



rishiraj5ed920846a38d002b73be4af  
rishiraj.gupta2018@vitstudent.ac.in  
8895279352  
1234567890  
Thu Jun 04 2020 16:25:40 GMT+0000 (Coordinated Universal Time)

AUTHORIZEREMOVE



Press confirm to authorize!

Username

rishiraj5ed920846a38d002b73be4af5ed920846a38d002b73be4af

Adhaar

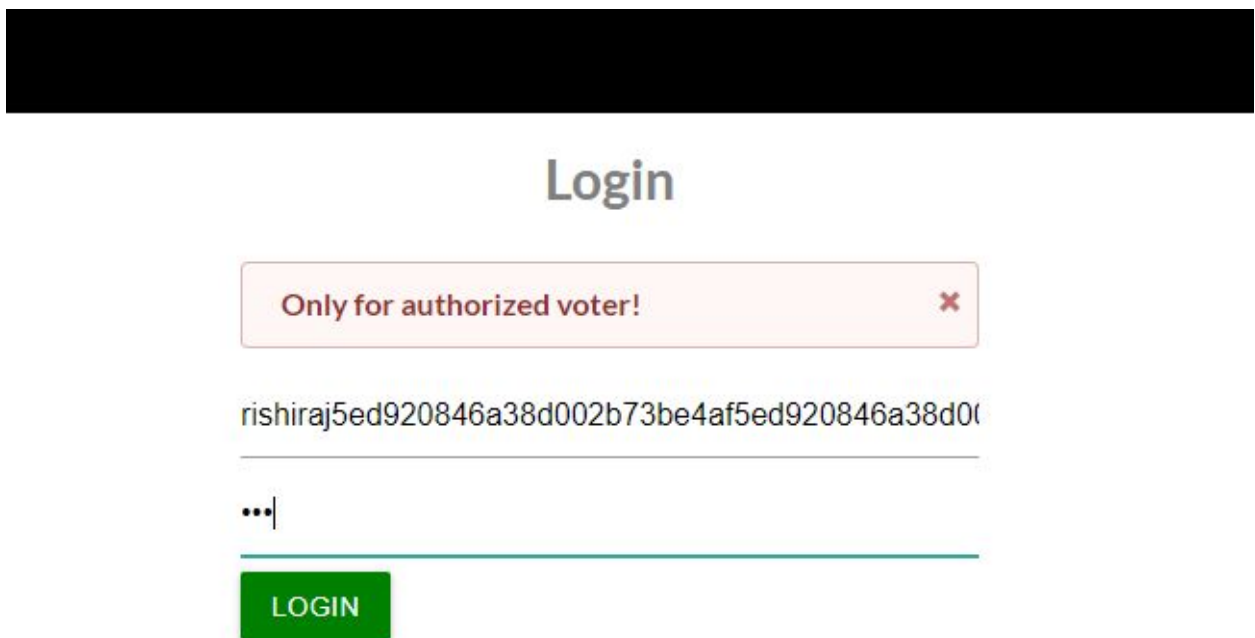
1234567890

Confirm

The authorization process will be confirmed with an encrypted username which is unique for every user that will be send to the

user via email or sms. The user will now be categorized as authorized user.

As shown the user authorization with an encrypted username, the username is applied in the login of the user on the date of the voting from where the user can only login through the provided username.

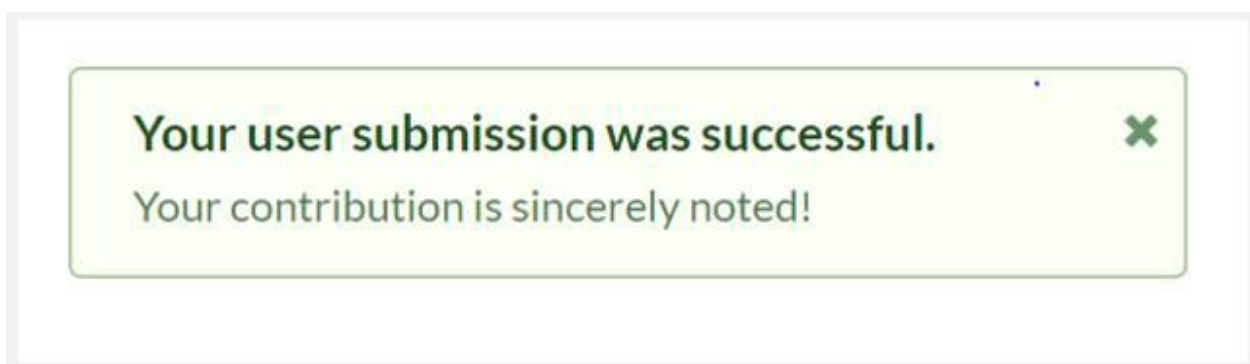


The image shows a login form with a black header bar at the top. Below the header, the word "Login" is centered in a large, bold, grey font. Underneath "Login" is a red-bordered box containing the text "Only for authorized voter!" in red, with a red 'x' icon to its right. Below this box is a text input field containing the encrypted username "rishiraj5ed920846a38d002b73be4af5ed920846a38d0f". Below the input field is a green "LOGIN" button. The form is set against a white background.

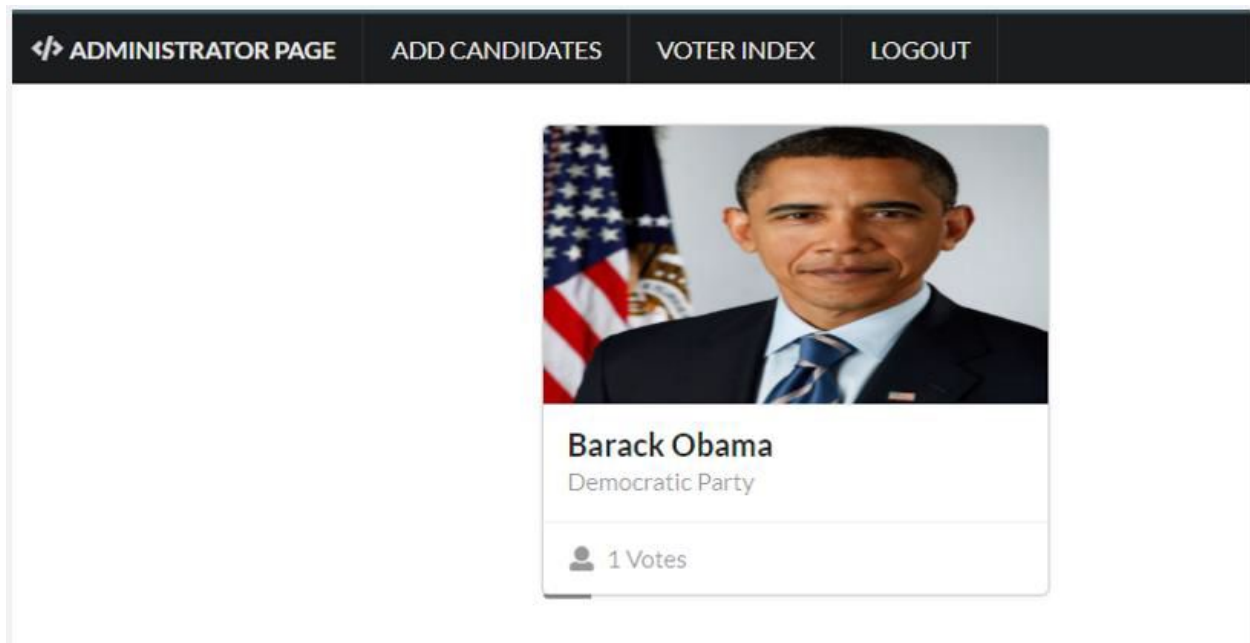
After the user logs in he/she will be directed to the voting page where the candidate details are available with a voting button.



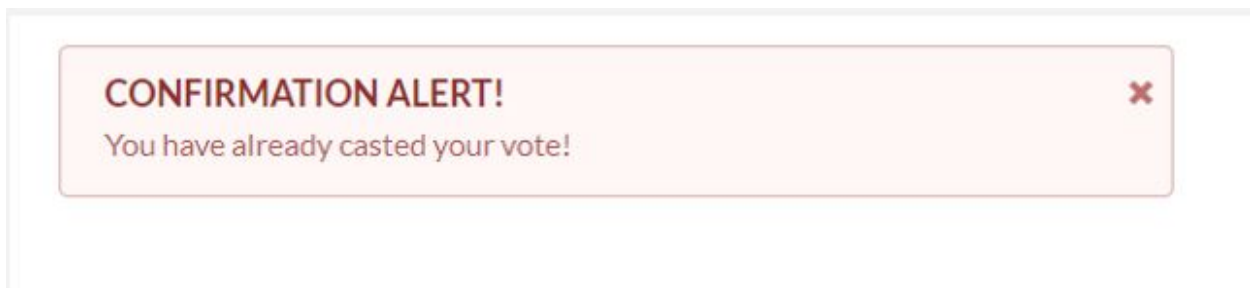
A confirmation page will be generated to ensure the vote passes on to the desired candidate.



After the voter confirms his choice as depicted he will be redirected to a success page showing the confirmation and commitment of the choice in the user database.



The admin can view the results of the polling in his page.



If the voter again tries to cast vote he/she will be directed to a warning page instead of success page which will indicate that the casted vote is duplicated and will not be saved in the database.

## SAMPLE CODE

```
var express =
require('express'); var
app = express();

var session =
require('express-session'); var
passport = require('passport');

var LocalStrategy =
require('passport-local'); var
mongo = require('mongodb');

var
bodyParser=require('body-pa
rser'); var flash =
require('connect-flash');

var
mongoose=require('mongoos
e'); var
shortID=require('short-mong
o-id');
```

```
var methodOverride=require("method-override");

var passportLocalMongoose =
require("passport-local-mongoose"); var
User=require("./public/schema/userschema");

var  expressSanitizer =
require('express-sanitizer'); var
cookieParser = require('cookie-parser')

var AdminSchema = new
mongoose.Schema({ cname:String,

    cage:N
    umber,
    cparty:
    String,
    cinfo:S
    tring,
    cimage
    :String,
    ccount: {type:
    Number,default:0},
    created: {type: Date,default:
    Date.now}
```



```
});
```

```
var admin = mongoose.model("Admin",AdminSchema);
```

#### **//APP CONFIG**

```
mongoose.connect("mongodb://localhost/voting"
); app.set("view engine","ejs");
app.use(express.static("public"));
app.use(bodyParser.urlencoded( {extended:true}))
; app.use(expressSanitizer());
app.use(methodOverride("_method"));
app.use(cookieParser('secret'));
```

#### **// PASSPORT CONFIGURATION**

```
app.use(require("express-sessi
")
({ secret:"Something!!",resave:
false, saveUninitialized: false
}));
app.use(flash());
app.use(passport.initialize(
));
app.use(passport.session())
; passport.use(new
LocalStrategy(User.authen
ticate()));
```



```
passport.serializeUser(Use
r.serializeUser());
passport.deserializeUser(U
ser.deserializeUser());
```

```
app.use(function(req, res, next){
  res.locals.currentUser = req.user;
  res.locals.success =
  req.flash('success'); res.locals.error =
  req.flash('error');
  next();
});
```

#### **//ROOT ROUTE**

```
app.get("/",functi
on(req,res){
  res.render("home
");
});
```

#### **//ADD new candidates**

```
app.get("/admin/new",functio
n(req,res){ res.render("add");
});
```

#### **//ADMIN PAGE**

```
app.get("/admin",function(req,res){
  res.render("admin");
});
```

#### **//Create**

```
app.post("/admin",function(req,res){
  admin.create(req.body.admin,function(err,newCandidate)
  {
    if(err){
      res.render("add");
    } else
  });
});

res.redirect("/admin");
```

#### **//VOTER INDEX**

```
app.get("/indexvoter",function(req,
res){
  User.find({},function(err,voterinde
x){ if(err){  });
});
}else{ } console.log("Error!");

res.render("voterindex",{voters:voterindex});
```

  
**//VOTER EDITING AND ENCRYPTING route**

```
app.get("/voterindex/:id/edit",function(req,res){  
  User.findById(req.params.id,function(err,Voters){
```

```
    if(err){  
      }); });
```

```
    }else{  
  } res.redirect("/admin");
```

```
  res.render("editvoter",{voter:  Voters})
```

**//UPDATE ROUTE**

```
app.put("/voterindex/:id",function(req,res){  
  User.findByIdAndUpdate(req.params.id,req.body.voter,function(err,updatedVoter)  
  {
```

```
    if(err){  
      }else{
```

```
  res.redirect("/admin");
```

```
    res.redirect("/indexvoter");
```

```
    });
```

```
  });
```

## **//DELETE VOTER**

```
app.delete("/voterindex/:id",function(req,res){
  User.findByIdAndRemove(req.params.id,function(err
  ){

    if(err){ });
  });
} else{ }

res.redirect("/indexvoter");
res.redirect("/indexvoter");
```

## **//Vote Count**

```
app.get("/votecount",function(req,res){
  admin.find( {},function(err,candidates) {}

  console.log(err);

  console.log(candidates);

  res.render("votecount",{candidates:candidates});
```

## **//vote route**

```
app.get("/vote",function(req,res){

  admin.find( {},function(err,candidate){

    if(err){ }); });

  } else{ }
```

```

console.log("Error!");

res.render("votepage",{candidates:candidate});

console.log(candidate);

app.get("/candidate/:id",function(req,res){
admin.findById(req.params.id,function(err,candidates){

if(err){

    });

    });

}else{

}console.log("Error!");

res.render("confirmcan",{candidate:candidates});

```

#### **//VOTING LOGIC**

```

app.post("/voter/:id",function(req,res){
User.findById(req.user._id,function(err,user){

    if(err){

        console.log(err);

        admin.findById(req.params.id,function(err,foundcandidate){    if(err){

res.render("votepage");}else {

```

```
if(user.candidate!==undefined){
res.render("warningage");

}else{

console.log("posted!!"); console.log(user);

console.log(foundcandidate);

console.log("done!");

user.candidate=foundcandidate.cname;

user.save(); foundcandidate.ccount+=1;

foundcandidate.save(function(err){
if(err){

    console.log(err);

    }

    })

    res.render("successage",{candidate:foundcandidate});

});

});

    }

});
```

# ALGORITHMS

In MongoDB, an operation on a single document is atomic. Because you can use embedded documents and arrays to capture relationships between data in a single document structure instead of normalizing across multiple documents and collections, this single-document atomicity obviates the need for multi-document transactions for many practical use cases.

For situations that require atomicity of reads and writes to multiple documents (in a single or multiple collections), MongoDB supports multi-document transactions. With distributed transactions, transactions can be used across multiple operations, collections, databases, documents, and shards.

## MongoDB molecular similarity map-reduce query

For code and explanation: <https://dzone.com/articles/joy-algorithms-and-nosql>

Other CODE:

/\*

For a replica set, include the replica set name and a seedlist of the members in the URI string; e.g.

String uri = "mongodb://mongodb0.example.com:27017,mongodb1.example.com:27017/admin?replicaSet=myRepl";

For a sharded cluster, connect to the mongos instances; e.g.

String uri = "mongodb://mongos0.example.com:27017,mongos1.example.com:27017:27017/admin";

\*/

```
client.getDatabase("mydb1").getCollection("foo")
```

```
    .withWriteConcern(WriteConcern.MAJORITY).insertOne(new Document("abc", 0));
```

```

client.getDatabase("mydb2").getCollection("bar")

    .withWriteConcern(WriteConcern.MAJORITY).insertOne(new Document("xyz", 0));

/* Step 1: Start a client session. */

final ClientSession clientSession = client.startSession();

/: Define the sequence of operations to perform inside the transactions. */

TransactionBody txnBody = new TransactionBody<String>() {

    public String execute() {

        MongoCollection<Document> coll1 = client.getDatabase("mydb1").getCollection("foo");

        MongoCollection<Document> coll2 = client.getDatabase("mydb2").getCollection("bar");

        /*

            Important:: You must pass the session to the operations.

        */

        coll1.insertOne(clientSession, new Document("abc", 1));

        coll2.insertOne(clientSession, new Document("xyz", 999));

        return "Inserted into collections in different databases";

    }

};

try {

    /*

        : Use .withTransaction() to start a transaction,


        execute the callback, and commit (or abort on error).

        */ clientSession.withTransaction(txnBody, txnOptions);

    } catch (RuntimeException e) {

```



```
  
  
    // some error handling  
  
} finally {  
  
    clientSession.close();  
  
}
```

## Use ObjectID as a **unique identifier**

**MongoDB database drivers by default generate an ObjectID identifier that is assigned to the `_id` field of each document. In many cases the ObjectID may be used as a unique identifier in an application.**

## Transactions and Operations

Distributed transactions can be used across multiple operations, collections, databases, documents, and, starting in MongoDB 4.2, shards.

```
db.coll.aggregate([  
  
  { $group: { _id: null, distinctValues: { $addToSet: "$x" } } },  
  
  { $project: { _id: 0 } }  
  
])
```

```
db.coll.aggregate([  
  
  { $match: { status: "A" } },  
  
  { $group: { _id: null, distinctValues: { $addToSet: "$x" } } },  
  
  { $project: { _id: 0 } }  
  
])
```

Source mongodb

MongoDB utilizes the Advanced Encryption Standard (AES) 256-bit encryption algorithm, an encryption cipher which uses the same secret key to encrypt and decrypt data

Weblink for

more: <https://docs.mongodb.com/manual/core/security-client-side-encryption/>

## AES256-GCM and Filesystem Backups

For [encrypted storage engines](#) that use AES256-GCM encryption mode, AES256-GCM requires that every process use a unique counter block value with the key.

For [encrypted storage engine](#) configured with AES256-GCM cipher:

- **Restoring from Hot Backup**

Starting in 4.2, if you restore from files taken via “hot” backup (i.e. the `mongod` is running), MongoDB can detect “dirty” keys on startup and automatically rollover the database key to avoid IV (Initialization Vector) reuse.

- **Restoring from Cold Backup**

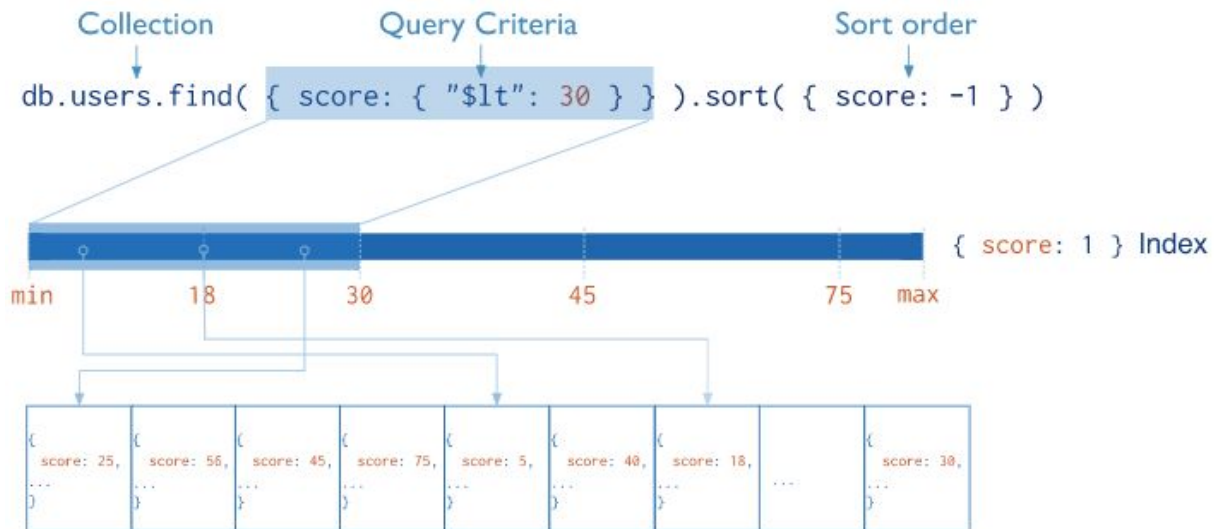
However, if you restore from files taken via “cold” backup (i.e. the `mongod` is not running), MongoDB cannot detect “dirty” keys on startup, and reuse of IV voids confidentiality and integrity guarantees.

Starting in 4.2, to avoid the reuse of the keys after restoring from a cold filesystem snapshot, MongoDB adds a new command-line option

`--eseDatabaseKeyRollover`. When started with the `--eseDatabaseKeyRollover` option, the `mongod` instance rolls over the database keys configured with AES256-GCM cipher and exits.

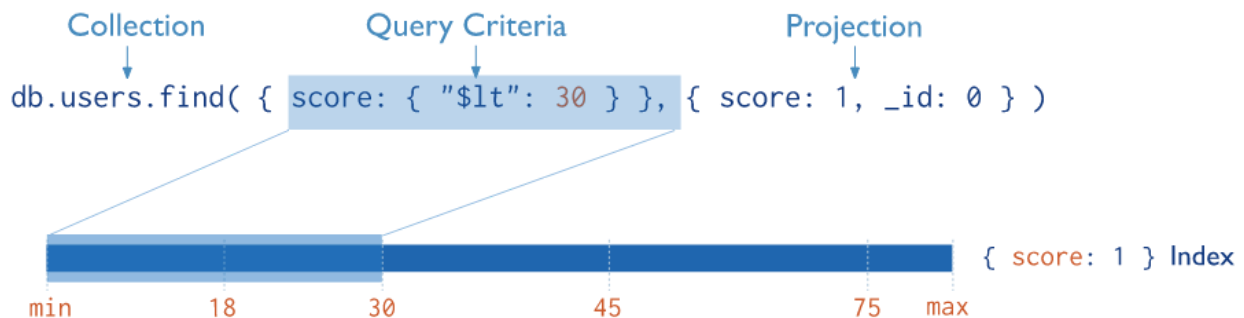
TIP

- In general, if using filesystem based backups for MongoDB Enterprise 4.2+, use the “hot” backup feature, if possible.
- For MongoDB Enterprise versions 4.0 and earlier, if you use AES256-GCM encryption mode, do **not** make copies of your data files or restore from filesystem snapshots (“hot” or “cold”).



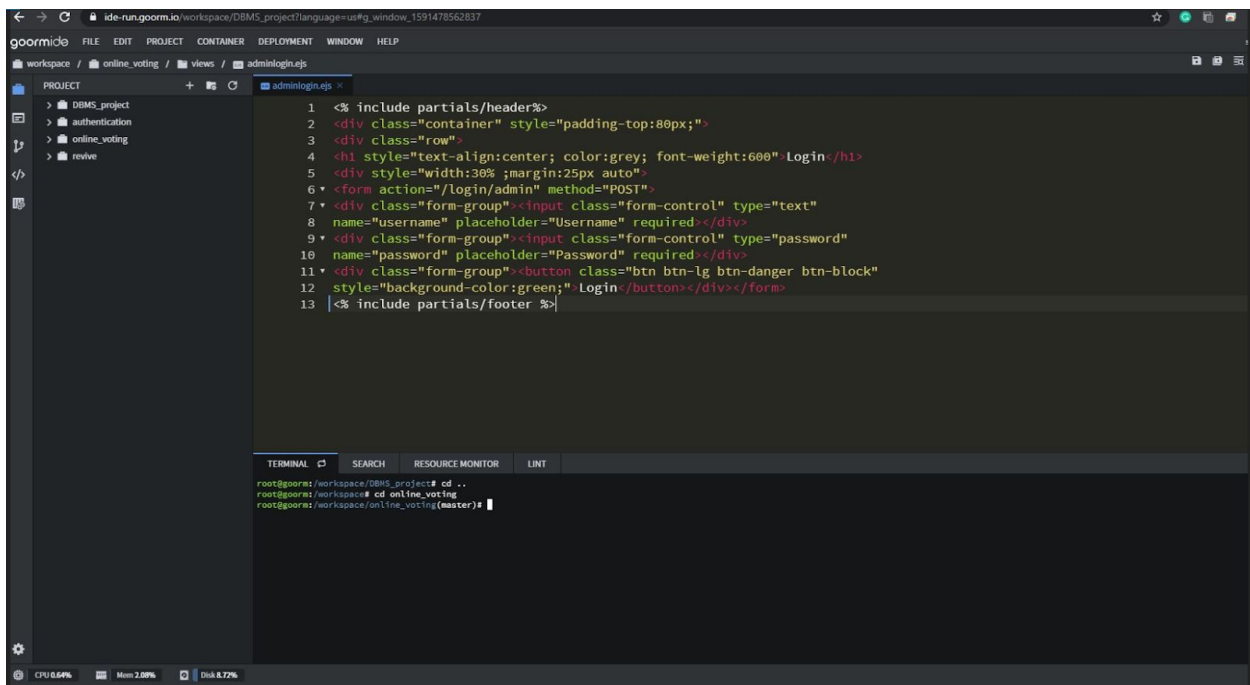
## Covered Queries

When the query criteria and the [projection](#) of a query include *only* the indexed fields, MongoDB returns results directly from the index *without* scanning any documents or bringing documents into memory. These covered queries can be very efficient.



## Online IDE results :

### 1:front end initiation(html,css,and node.js ,etc)



The screenshot displays the goorm.io online IDE interface. The top menu bar includes options like FILE, EDIT, PROJECT, CONTAINER, DEPLOYMENT, WINDOW, and HELP. The left sidebar shows a project tree with folders for DBMS\_project, authentication, online\_voting, and revive. The main editor area shows a file named adminlogin.js with the following HTML code:

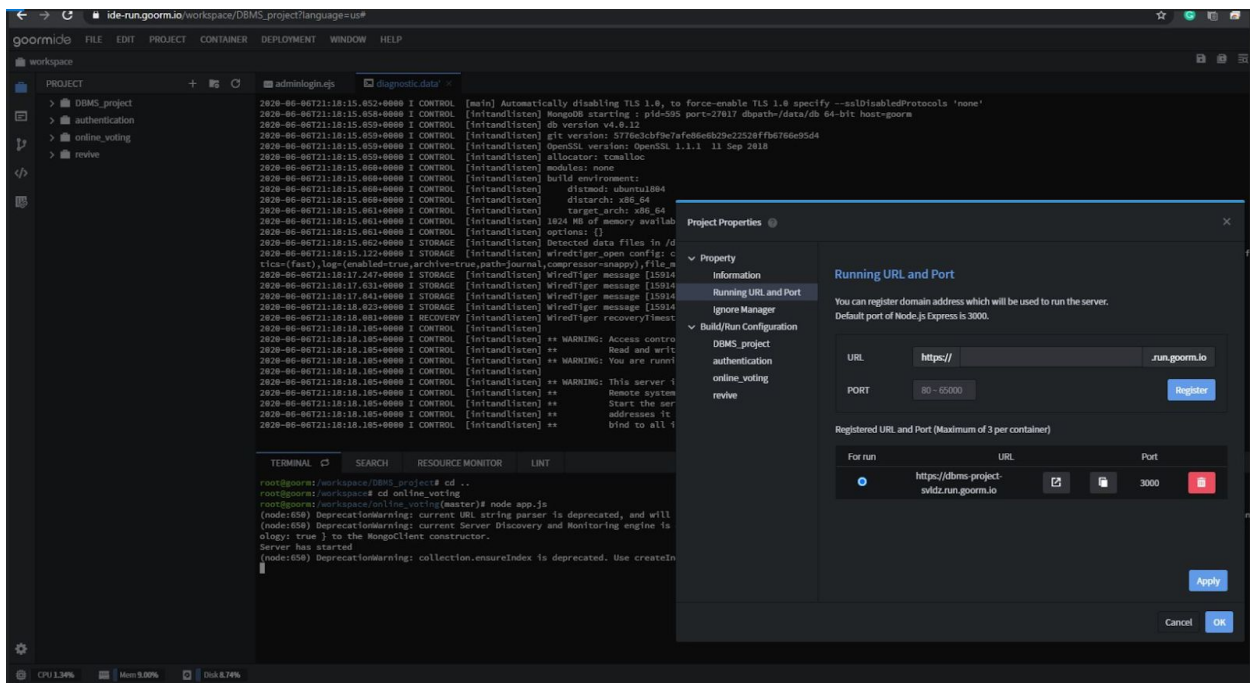
```
1 <% include partials/header%>
2 <div class="container" style="padding-top:80px;">
3 <div class="row">
4 <h1 style="text-align:center; color:grey; font-weight:600">Login</h1>
5 <div style="width:30%; margin:25px auto">
6 <form action="/login/admin" method="POST">
7 <div class="form-group"><input class="form-control" type="text"
8 name="username" placeholder="Username" required</div>
9 <div class="form-group"><input class="form-control" type="password"
10 name="password" placeholder="Password" required</div>
11 <div class="form-group"><button class="btn btn-lg btn-danger btn-block"
12 style="background-color:green;">Login</button></div></form>
13 <% include partials/footer %>
```

Below the code editor, there is a terminal window showing the following commands and output:

```
root@goorm:/workspace/DBMS_project# cd ..
root@goorm:/workspace# cd online_voting
root@goorm:/workspace/online_voting(master)#
```

The bottom status bar indicates system resources: CPU 0.64%, Mem 2.00%, and Disk 8.72%.

## 2:back end initiation (mongo db ,connection, url generation,etc)



---

## CONCLUSION

This Online Voting system will manage the Voter's information by which voter can login and use his voting rights. The system will incorporate all features of Voting system. It provides the tools for maintaining voter's vote to every party and it count total no. of votes of every party. There is a DATABASE which is maintained by the ELECTION COMMISSION OF INDIA in which all the names of voter with complete information is stored

## REFERENCES

<http://scihub.tw/https://ieeexplore.ieee.org/document/8276077>

<https://ieeexplore.ieee.org/document/8276077>

<https://www.ijircst.org/DOC/10f175cd9870e54b3fb4d5-a3c6ea0f38bb.pdf>

—