DemoDB ->
mongodb+srv://admin:zWmghAsy49UmZQ39@cluster0.qvsfl.mongodb.net/?retryWrites=true&w=majority
MongoDB Atlas →

MongoDB Atlas website → Login to Ur Google Acount → Top Left Corner One Dropdown (Select New Project) → Enter Project Name → click next → click create project → Click Build Cluster → Click Create cluster **FREE** → Click AWS → Select Asian Mumbai server → Click Bottom Button to Create Cluster → DB Creating It Takes Someting →

Left side bar → click Database Access → click Add New Database Add User → Enter Username And Password → provide privileges → click Add User →

Left side bar → click Network Access → click Add Ip Address → select Allow Access form any where → click confirm

Left Side bar → click Cluster → DB will Be created → Click Connect → Click Connect your application → Copy The MongoDB URL → Replace the password ur username and password → and used in Node.js

--

Mongoodb pass → admin, 2w050tEV2N0rqhh7

mongodb+srv://admin:<password>@cluster0.jn78j.mongodb.net/myFirstDatabase?retryWrites=true&w=majority

|-> the <password> should be replaced by user password of db in the above line.

====================== Node.js Start ===================================

```
mkdir myapp
cd myapp

npm init

npm install express

npm install eslint --save-dev
```

Installing the Express Application Generator

```
npm install express-generator -g
```

express helloworld

cd helloworld
npm install

Create Folder:

create folder → config
    Create file → config/config.js

config.js

```
const mongoose = require('mongoose');

var env = require('dotenv').config()

mongoose.connect(process.env.DBURL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    useFindAndModify: true,
},
    (err) => {
        if (!err) {
            console.log('MongoDB Connection Succeeded.')
        } else {
            console.log('Error in DB connection : ' + err)
        }
    });



require('../models/index')
```

Require config in app.js

```
// global require mongoose
global.mongoose = require('mongoose');
global.config = require('./config/config');
```

—---

Create folder → controller

Create file → controller/collection.js & index.js

Collection.js

```
const branch = mongoose.model("branch");
module.exports = {
    branchDropdown(req, res, next) {
        branch.find({}).then((value) => {
            return res.status(200).json({
                        status: true,
                        message: "Record Found...",
                        data: value
                    })
      }).catch((err) => {
            res.status(200).json({
                status: false,
                message: err.message,
            })
        })
}
```

Index.js

```
const branch = require('../controller/settings/branch');

module.exports = {
    branch,
}
```

—--

Create folder → models

Create file → models/attribute.js & index.js

attribute.js

```
const mongoose = require('mongoose');
const timestamps = require('mongoose-timestamp');
const ObjectId = require('mongodb').ObjectID;

var itemcatarr = new mongoose.Schema({
    itemcategory: {
        type: ObjectId,
    },
})

const schema = new mongoose.Schema({
    attribute: {
        type: String,

    },
    attributetype: {
        type: String,
    },
    categories: {
        type: [{ type : ObjectId, required: true }],
    },
    itemcategory: [itemcatarr],
    prefix: {
        type: String,
    },
    numberformat: {
        type: String,
    },
    dateformat: {
        type: String,
    },
    required: {
        type: Boolean,
        default:false
    },
    displayinprint: {
        type: Boolean,
        default:false
    },
    dropdownvalues: {
        type: Array,
    },
    createdby: {
        type: ObjectId,
    },
    updatedby: {
        type: ObjectId
    },
    isdeleted: {
        type: Number,
        default: 0
    },
})


schema.plugin(timestamps);

const attribute = new mongoose.model('attribute', schema);

module.exports = attribute;
```

Index.js

```
require('../models/master/attribute');
```

—---

Routes:

```

// ~ account
const account = require('../controller').account
router.get('/accountgroupparentlist', account.accountgroupparentlist)





// function verifytoken(req, res, next) {
//     try {
//         var token = req.headers.authorization
//        // console.log(token);
//         if (token == undefined) {
//             return res.status(200).json({
```

```
//                 "status": false,
//                 "message": `Please Send Token...`
//             })
//         }
//         jwt.verify(token,process.env.JWTSECKEY, function(err,
decoded) {
//             if (!decoded) {
//                 return res.status(200).json({
//                     "status": false,
//                     "message": `Invalid Token....`
//                 })
//             } else {
//                 next()
//             }
//         });
//     } catch (error) {
//         return res.status(200).json({
//             "status": false,
//             "message": error.message
//         })
//     }

// }
```

—---

.env
DBURL=mongodb+srv://zims_dev:KsMmfHfZNTtJ2KLZ@cluster0.fzsyi.mongodb.net/zims_dev?retryWrites=true&w=majority

PORT=6377

JWTSECKEY=iehfiebfieff654f984f14f6541yggfmnxbjwqudbfibhdiehfiebfieff654f984f14f6541yggfmnxbjwqudbfibhdshb1tf67tbhjhrgibr78majquirolcbndgtwb653bhydbhkasfh445shb1tf67tbhjhrgibr78majquirolcbndgtwb653bhydbhkasfh445

—---

Alter File:

bin → www.js

www:

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('myapp:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '5055');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port,()=>{
  console.log(`server listen on port
http://localhost:${process.env.PORT}/`);
});
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);
  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
```

```
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

—---

File Upload:

```
var multer = require('multer')

var itemstorage = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, './public/billing/item')
    },
    filename: function(req, file, cb) {
        cb(null, "item" + Date.now() + file.originalname)
    }
})
var upload = multer({ storage: itemstorage })
router.post('/item', upload.single("item_img"), (req, res) => {
    return res.status(200).json(req.file.filename)
})
```

—---

Global Require:

```
// global require mongoose
global.mongoose = require('mongoose');
global.config = require('./config/config');
```

—---

app.js
—--------

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
// cors
var cors = require('cors');

// global require mongoose
global.mongoose = require('mongoose');
global.config = require('./config/config');

var app = express();

//cors
app.use(cors())


app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);

app.use('/admin', adminRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
    next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
    // set locals, only providing error in development
    res.locals.message = err.message;
    res.locals.error = req.app.get('env') === 'development' ? err : {};
    // render the error page
    res.status(err.status || 500);
    res.render('error');
});

module.exports = app;
```

—---

========================= Node.js End ==================================

Npm install dotenv

MODEL →

```
const mongoose = require('mongoose');
var timestamps = require('mongoose-timestamp');
var ObjectId = require('mongodb').ObjectID;
var otherchargeschema = new mongoose.Schema({
    date: {
        type: Date,
    },
    reason: {
        type: String,
    },
    amount: {
        type: Number,
    },
    desc: {
        type: String,
    },
    isdeleted: {
        type: Number,
        default: 0
    },
    isactive: {
        type: Boolean,
        default: true
    },
})
var schema = new mongoose.Schema({
    paymentid:{
        type: String,
    },
    patientid: {
        type: ObjectId,
    },
    patientname: {
        type: String,
    },
    mobile: {
        type: String,
    },
    standardcharges: {
        type: String,
    },
    appliedcharges: {
        type: String,
    },
    othercharges: {
        type: [otherchargeschema],
    },
    paymode: {
        type: String,
    },
    description: {
        type: String,
    },
    isdeleted: {
        type: Number,
        default: 0
    },
    isactive: {
        type: Boolean,
        default: true
    },
    createdby: {
        type: ObjectId,
    },
})

schema.plugin(timestamps);

var payment = new mongoose.model('payment', schema);
```

```
module.exports = payment;
```

Controller →

```
const mongoose = require('mongoose');
var ObjectId = require('mongodb').ObjectID;
const modelschema = mongoose.model("holiday");

module.exports = {
    insert(req, res) {
        if (req.body._id) {
            modelschema.find({
                "_id": req.body._id
            }).then((data) => {
                if (data.length == 0) {
                    res.status(200).json({
                        "status": false,
                        "message": "No Record Found......"
                    })
                } else {
                    data[0].update({
                        holidaytype: req.body.holidaytype,
                        holidaydate: req.body.holidaydate,
                        name: req.body.name,
                        description: req.body.description,
                    }).then((data) => {
                        res.status(200).json({
                            "status": true,
                            "message": "record update sucessfull..."
                        })
                    })
                }
            })
        }
        else {
            modelschema.find({
                holidaytype: req.body.holidaytype,
                holidaydate: req.body.holidaydate,
                name: req.body.name,
                isdeleted: 0,
                isactive: true
            }).then((result) => {
                if (result.length == 0) {
                    {
                        return modelschema.create({
                            holidaytype: req.body.holidaytype,
                            holidaydate: req.body.holidaydate,
                            name: req.body.name,
                            description: req.body.description,
                            createdby: req.body.createdby,
                        }).then((result) => {
                            return res.status(200).json({
                                "status": true,
                                "message": "record add sucessfull..."
                            })
                        }).catch((err) => {
                            return res.status(200).json({
                                "status": false,
                                "message": err.message
                            })
                        });
                    }
                }
                else {
                    return res.status(200).json({
                        "status": false,
                        "message": `${req.body.name} already add this
hospital...`
                    })
                }

            }).catch((err) => {
                return res.status(200).json({
                    "status": false,
                    "message": err.message
                })
            });
        }

    },
    list(req, res) {
        modelschema.aggregate([{
            $match: {
                isdeleted: 0,
                isactive: true
            }
        },


        {
            $sort: {
                _id: -1
            }
        }]).then((result) => {
            if (result.length == 0) {
                return res.status(200).json({
                    "status": false,
                    "message": "No Record Found..."
                })
            }
            else {
                return res.status(200).json({
                    "status": true,
                    "list": result
                })
            }
        }).catch((err) => {
            return res.status(200).json({
                "status": false,
                "message": err.message
            })
        });
    },
    delete(req, res) {
        modelschema.find({
            "_id": req.body._id
        }).then((data) => {
            if (data.length == 0) {
                res.status(200).json({
                    "status": false,
                    "message": "No Record Found......"
                })
            } else {
                data[0].update({
                    isdeleted: 1,
                }).then((data) => {
                    res.status(200).json({
                        "status": true,
                        "message": "Deleted Sucessfully....."
                    })
                })
            }
        })
    },
}
```

App.js →

```
// router
var indexRouter = require('./routes/index');
var adminRouter = require('./routes/admin');
var masterRouter = require('./routes/master');
```

```
var app = express();

//cors
app.use(cors())
```

```
app.use('/', indexRouter);

app.use('/admin', adminRouter);

app.use('/hospital/master', masterRouter);
```

Router →

```


// ---------------------------with token
verify--------------------------------//



// function verifytoken(req, res, next) {
```

```
//      try {
//          var token = req.headers.authorization
//         // console.log(token);
//          if (token == undefined) {
//              return res.status(200).json({
//                  "status": false,
//                  "message": `Please Send Token...`
//              })
//          }
//          jwt.verify(token,process.env.JWTSECKEY, function(err,
decoded) {
//              if (!decoded) {
//                  return res.status(200).json({
//                      "status": false,
//                      "message": `Invalid Token....`
//                  })
//              } else {
//                  next()
//              }
//          });
//      } catch (error) {
//          return res.status(200).json({
//              "status": false,
//              "message": error.message
//          })
//      }

// }

// ---------------------------with out token
verify-------------------------------//

function verifytoken(req, res, next) {
    next()

}
```

```
const hospital = require('../controller').hospital
router.post('/hospital/add', verifytoken, hospital.insert)
router.get('/hospital/list', verifytoken, hospital.list)
router.post('/hospital/delete', verifytoken, hospital.delete)
```

Image Upload →

```

var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, './public/document')
  },
  filename: function (req, file, cb) {
    cb(null, "patientdocument" + Date.now() + file.originalname)
}
})

var upload = multer({ storage: storage })

router.post('/patientdocument', upload.single("patientpdfdocument"),
(req, res) => {

  return res.status(200).json(req.file)
})

```

Config.js →

```
const mongoose = require('mongoose');

var env = require('dotenv').config()

mongoose.connect(process.env.DBURL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    useFindAndModify: true,
},
    (err) => {
        if (!err) {
            console.log('MongoDB Connection Succeeded.')
        } else {
            console.log('Error in DB connection : ' + err)
        }
    });

```

```
require('../models/index')
```

WWW →

server.listen(port,()=>{
  console.log(`server listen on port  http://localhost:${process.env.PORT}/`);
});
server.on('error', onError);
server.on('listening', onListening);

Gitignore →

```
# Node artifact files
node_modules/
dist/
public/
```

==

Multiple search | optional search | optional find | multiple find | $in operator |

```
                                accountledger.find({
                                    'accountledger': { $in: [
                                        ledgers.purchase,
                                        ledgers.discount,
                                        ledgers.lessDiscount,
                                        ledgers.roundOff,
                                        // ledgers.roundOffAdd,
                                        // ledgers.roundOffMinus
                                    ]},
                                    isdeleted: 0
                                })
```

—---

Login Validation:

Add:

```
const personAdd = async ({body = {}}) => {
    const {name, age, password: praw } = body
    const password = await hashGenerator(praw)
    const addPerson = await person.create({
        name,
        age,
        password,
    }).catch(err => catchError(err))
    if(addPerson) return responseObject(200, true, 'Person added
successfullly', addPerson)
}
```

Signin:

```
 const signin = async ({body = {}}) => {
     const {name, password} = body
     const checkUser = await person.findOne({name})
     if(checkUser) {
         const {password: hashedPassword} = checkUser
         const validUser = await hashValidator(password,
hashedPassword)
         if(validUser) {
             return await responseObject(200, true, 'Login
Successfully', checkUser)
            } else {
                return await responseObject(200, false, 'Invalid
Password')
            }
     }
     if(!checkUser) return await responseObject(200, false, 'Invalid
Username')
 }
```

Set JWT:

```
const tokenGenerator = (name) => {
    const token = jwt.sign(
        { name },
        process.env.JWT_KEY,
        { expiresIn: '3hours' }
        )
    return token
}
```

```
 const signin = async ({body = {}}, res) => {
     const {name, password} = body
     const checkUser = await person.findOne({name})
     if(checkUser) {
         const {password: hashedPassword} = checkUser
         const validUser = await hashValidator(password,
hashedPassword)
         if(validUser) {
            const token = await tokenGenerator(name)
             res.cookie('jwt', token)    // * set JWT // * using
cookie-parser
             return await res.send(token)
            } else {
                return await res.send('Invalid Password')
            }
     }
     if(!checkUser) return await res.send('Invalid Username')
 }
```

Protected Routes:

```
const tokenValidator = async (token) => {
    try {
        const data = jwt.verify(token, process.env.JWT_KEY);
        return data;
    }
    catch(err) {
        return false
    }
}
```

```
const jwtAuthentication = async (req, res, next) => {
    try {
        const { jwt } = req.cookies;
        const valid = await tokenValidator(jwt)
        if(valid) {
            next()
        } else {
            return res.status(200).send('Invalid Access')
        }
    }
    catch(err) {
        return res.status(200).send('Invalid Access')
    }
}

router.get('/protected',jwtAuthentication, (req, res) =>
res.status(200).send('after login'))
```

——

Duplicate UserName | Duplicate ID | check if _id is duplicate | duplicate _id | check _id is duplicate | check _id is already exist | check if _id already exist

```
const addUser = async ({body = {}}) => {
    const { userName, email, mobile, age } = body
    const checkDuplicateUserName = await
user.find({userName}).lean().then(data => data).catch(err =>
error.catchError(err))
    if(checkDuplicateUserName.error)  return
checkDuplicateUserName.error
    if(checkDuplicateUserName.length) return error.error(400, false,
`${userName} Already Exist`)
    if(!checkDuplicateUserName.length) {
        const addData = await user.create({
            userName,
            email,
            mobile,
            age,
        }).then(data => data).catch(err => error.catchError(err))
        if(addData.error) return addData.error
        if(!addData) return error.error(400, false, 'failed')
        if(addData) return response.responseObject(200, true, 'Added
Sucessfully')
```

```
    }
}
```

Declare array contain objectId | array declaration

```
const schema = new mongoose.Schema({
    courseId: {
        type: [
           { type: ObjectId, ref: courseSchema}
         ]
    },
    createdby: {
        type: ObjectId,
    },
    updatedby: [updatedby]
})
```

Get list

Service:

```

const userList = async () => {
    const list = await user.find({}).lean().then(data =>
data).catch(err => error.catchError(err))
    if(list.error) return list.error
    if(!list.length) return error.error(200, false, 'No Record Found')
    if(list.length) return response.responseObject(200, true, 'Record
Found', list)
}
```

Router:

```
router.get('/list', async (req, res) => {
    user.userList().then(({statusCode, status, message, data = null})
=>
        response.response(res, statusCode, status, message, data) )
})
```

Add user log | add data after add data | two add operation | userlog | add userlog | create userlog | add userlog after add data

```
```

```
const userLogAdd = async (userId, modelName, action) => {
    const addData = userlog.create({userId, modelName, action})
}

const addUser = async ({body = {}}) => {
    const { userName, email, mobile, age } = body
    const addData = await user.create({
        userName,
        email,
        mobile,
        age,
    }).then(data => data).catch(err => error.catchError(err))
        if(addData.error) return addData.error
        if(!addData) return error.error(400, false, 'failed')
        if(addData) {
            const { _id } = addData
            await userLogAdd(_id, 'User', 'Add Data')
            return await response.responseObject(200, true, 'Added
Sucessfully', addData)
        }
}
```

Use Promise all:

```
const userLogAdd = async (userId, modelName, action) => {
    const addData = userlog.create({userId, modelName, action})
}

const addUser = async ({body = {}}) => {
    const { userName, email, mobile, age } = body
    const addData = await user.create({
        userName,
        email,
        mobile,
        age,
    }).then(data => data).catch(err => error.catchError(err))
        if(addData.error) return addData.error
        if(!addData) return error.error(400, false, 'failed')
        if(addData) {
            const { _id } = addData
            await Promise.all([userLogAdd(_id, 'User', 'Add Data')])
            return await response.responseObject(200, true, 'Added
Sucessfully', addData)
```

```
        }
}
```

Match 2 objectId in array prototype | match two object Id in array prototype | Match 2 objectId in array method| match two object Id in array method

User `_id.equals(another_id)`

```
userDetails.find(({_id}) => _id.equals(userId))
```

List with $lookup | list with lookup | list with join table | list based on 2 table | lookup list | join list | join table list

Service:

```
const userLogListWithUserDetails = async () => {
    const list = await userlog.find({}).lean().then(data =>
data).catch(err => error.catchError(err))
    if(list.error) return list.error
    if(!list.length) return error.error(200, false, 'No Record Found')
    if(list.length) {
        const userDetails = await user.find({ _id: { $in: list.map(e =>
e.userId) }}).lean().then(data => data).catch(err =>
error.catchError(err))
        if(userDetails.error) return userDetails.error
        if(!userDetails.length) return error.error(400, false, 'No
Record Found')
        if(userDetails.length) {
            const result = await [...list.map((log) => {
                const {_id, userId, modelName, action, } = log
                const {userName} = userDetails.find(({_id}) =>
_id.equals(userId))
                return {_id, userId, userName, modelName, action}
            })]
            return await response.responseObject(200, true, 'Record
Found', result)
        }
    }
}
```

```
Routes:
```

```
router.get('/loglist', async (req, res) => {
    user.userLogListWithUserDetails().then(({statusCode, status,
message, data = null}) =>
        response.response(res, statusCode, status, message, data) )
})
```

MongoDB | aggregate filter | aggregate or | aggregate $or | find array value | array to value find | match value to array | match array value | array match

```
            let supplierarr = [];
            req.body.supplier.map(ele => {
                supplierarr.push({ 'supplier': ObjectId(ele) })
            })
            Object.assign(matchdata, { '$or': supplierarr })
```

MongoDB | find today | match current date | match today date | aggregate match current date | aggregate match today date | date match | date filter

```
                const startToDay = new Date(new Date().setUTCHours(0,
0, 0, 0))
                const endToDay = new Date(new Date().setUTCHours(23,
59, 59, 999))
                Object.assign(matchdata, { 'bill_date': {
                    '$gte': startToDay,
                    '$lt': endToDay
                  } })
```

MongoDB | match yesterday | date match

```
                let yesterday = new Date();
                yesterday.setDate(yesterday.getDate() - 1);
                const startYesterday = new
Date(yesterday.setUTCHours(0, 0, 0, 0))
```

```
                const endYesterday = new Date(yesterday.setUTCHours(23,
59, 59, 999))
                Object.assign(matchdata, { 'bill_date': {
                    '$gte': startYesterday,
                    '$lt': endYesterday
                  } })
```

MongoDB | match current week | match currentWeek | date match

```
                var curr = new Date; // get current date
                var first = curr.getDate() - curr.getDay(); // First
day is the day of the month - the day of the week
                var last = first + 6; // last day is the first day + 6
                var firstday = new Date(new
Date(curr.setDate(first)).setUTCHours(0, 0, 0, 0))
                var lastday = new Date(new
Date(curr.setDate(last)).setUTCHours(23, 59, 59, 999))
                Object.assign(matchdata, { 'bill_date': { $gte:
firstday, $lt: lastday } })
```

MongoDB | match lastWeek | match last week date | date match

```
                let lastWeekDate = new Date();
                lastWeekDate.setDate(lastWeekDate.getDate() - 7);
                let first = lastWeekDate.getDate() -
lastWeekDate.getDay();
                let last = first + 6; // last day is the first day + 6
                let firstday = new Date(new
Date(lastWeekDate.setDate(first)).setUTCHours(0, 0, 0, 0))
                let lastday = new Date(new
Date(lastWeekDate.setDate(last)).setUTCHours(23, 59, 59, 999))
                Object.assign(matchdata, { 'bill_date': { $gte:
firstday, $lt: lastday } })
```

MongoDB | match current Month | match current Month | date match

```
                      // d = new Date(); d.setFullYear(2008, 11, 0); //
Sun Nov 30 2008
```

```
                let date = new Date()
                let firstDayOfCurrentMonth = new Date(new Date(new
Date().setFullYear(date.getFullYear(), date.getMonth(),
1)).setUTCHours(0, 0, 0, 0))
                let lastDayOfCurrentMonth = new Date(new Date(new
Date().setFullYear(date.getFullYear(), date.getMonth()+1,
0)).setUTCHours(23, 59, 59, 999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': firstDayOfCurrentMonth,
                        '$lt': lastDayOfCurrentMonth
                    }
                })
```

MongoDB | match last Month | match lastMonth | date match

```
                let date = new Date()
                let firstDayOfLastMonth = new Date(new Date(new
Date().setFullYear(date.getFullYear(), date.getMonth()-1,
1)).setUTCHours(0, 0, 0, 0))
                let lastDayOfLastMonth = new Date(new Date(new
Date().setFullYear(date.getFullYear(), date.getMonth(),
0)).setUTCHours(23, 59, 59, 999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': firstDayOfLastMonth,
                        '$lt': lastDayOfLastMonth
                    }
                })
```

MongoDB | match current quarter year | date match

```
                let date = new Date()
                let octobarFirst = new Date(new Date(new
Date().setFullYear(date.getFullYear(), 9, 1)).setUTCHours(0, 0, 0, 0))
                let decemberFirst = new Date(new Date(new
Date().setFullYear(date.getFullYear(), 12, 0)).setUTCHours(23, 59, 59,
999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': octobarFirst,
                        '$lt': decemberFirst
                    }
```

MongoDB | match last quarter year | date match

```
                let date = new Date()
                let octobarFirst = new Date(new Date(new
Date().setFullYear(date.getFullYear()-1, 9, 1)).setUTCHours(0, 0, 0,
0))
                let decemberLast = new Date(new Date(new
Date().setFullYear(date.getFullYear()-1, 12, 0)).setUTCHours(23, 59,
59, 999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': octobarFirst,
                        '$lt': decemberLast
                    }
                })
```

MongoDB | match current financial year | match current fiscal year | date match

```
                let date = new Date()
                let januaryFirst = new Date(new Date(new
Date().setFullYear(date.getFullYear(), 0, 1)).setUTCHours(0, 0, 0, 0))
                let decemberLast = new Date(new Date(new
Date().setFullYear(date.getFullYear(), 12, 0)).setUTCHours(23, 59, 59,
999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': januaryFirst,
                        '$lt': decemberLast
                    }
                })
```

MongoDB | match last financial year | match last fiscal year | date match

```
                let date = new Date()
                let januaryFirst = new Date(new Date(new
Date().setFullYear(date.getFullYear()-1, 0, 1)).setUTCHours(0, 0, 0,
0))
                let decemberLast = new Date(new Date(new
Date().setFullYear(date.getFullYear()-1, 12, 0)).setUTCHours(23, 59,
59, 999))
                Object.assign(matchdata, {
                    'bill_date': {
                        '$gte': januaryFirst,
                        '$lt': decemberLast
                    }
                })
```

MongoDB | match from date | date match

```
        let date = new Date(req.body.fromDate)
        let from_date_startDay = new Date(new Date(new
Date().setDate(date.getDate())).setUTCHours(0, 0, 0, 0))
        let from_date_endDay = new Date(new Date(new
Date().setDate(date.getDate())).setUTCHours(23, 59, 59, 999))
        Object.assign(matchdata, {
            'bill_date': {
                '$gte': from_date_startDay,
                '$lt': from_date_endDay
            }
        })
```

MongoDB | match todate | date match

```
        let date = new Date(req.body.toDate)
        let from_date_startDay = new Date(new Date().setUTCHours(0, 0, 0, 0))
        let from_date_endDay = new Date(new Date(new
Date().setDate(date.getDate())).setUTCHours(23, 59, 59, 999))
        Object.assign(matchdata, {
            'bill_date': {
                '$gte': from_date_startDay,
                '$lt': from_date_endDay
            }
        })
```

MongoDB | match from date and to date | match fromDate and ToDate | acth fromDate & toDate | match from date & to date | date match

```
        let from_date = new Date(req.body.fromDate)
        let to_date = new Date(req.body.toDate)
        let from_date_startDay = new Date(new Date(new
Date().setDate(from_date.getDate())).setUTCHours(0, 0, 0, 0))
        let to_date_endDay = new Date(new Date(new
Date().setDate(to_date.getDate())).setUTCHours(23, 59, 59, 999))
        Object.assign(matchdata, {
            'bill_date': {
                '$gte': from_date_startDay,
                '$lt': to_date_endDay
            }
        })
```

Date Functions

```
const addDays = (days, date = new Date()) => {
    date = new Date(date)
    return new Date(date.setDate(new Date().getDate()+days))
}
const subtractDays = (days, date = new Date()) => {
    date = new Date(date)
    return new Date(date.setDate(new Date().getDate()-days))
}
const getStartDate = (date = new Date()) => {
    date = new Date(date)
    return new Date(new Date(date.setDate(new
Date().getDate()+1)).setHours(0,0,0,0))
}
const getEndDate = (date = new Date()) => {
    date = new Date(date)
    return new Date(date.setHours(23, 59, 59, 999))
}
```

Difference between 2 days | date difference

```
const dateDifference = (date1 = new Date(), date2 = new Date()) => {
    const diffTime = Math.abs(date2 - date1);
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
    return diffDays
}
```

Reduce array value to unique | remove duplicated in array | array duplicates | setUnion | remove duplicates | unique array

```
// demo array
        "attributes": [
            "61ab2f9dcb7a0c1cdc0b8109",
            "61aa04c7d01002376896ca5a",
            "61aa04c7d01002376896ca5a",
            "61ab2f9dcb7a0c1cdc0b8109",
            "61ab2fedcb7a0c1cdc0b810e",
            "61ab3035cb7a0c1cdc0b8114",
            "61ab3f908535952f108d0ed7",
            "61ab3017cb7a0c1cdc0b8112"
        ]
.
// code
```

```
attributes: {'$setUnion': ['$attributesArray', []]},
```

```
// result


        "attributes": [
            "61aa04c7d01002376896ca5a",
            "61ab2f9dcb7a0c1cdc0b8109",
            "61ab2fedcb7a0c1cdc0b810e",
            "61ab3017cb7a0c1cdc0b8112",
            "61ab3035cb7a0c1cdc0b8114",
            "61ab3f908535952f108d0ed7"
        ]
```
—---

Merge array | merge sub array | reduce array | merge array of array | reduce array of array | subarray

```
{
  $reduce: {
     input: [ [ 3, 4 ], [ 5, 6 ] ],
     initialValue: [ 1, 2 ],
     in: { $concatArrays : ["$$value", "$$this"] }
  }
}
```

```
[ 1, 2, 3, 4, 5, 6 ]
```

```
------------------------------------------------------------
```

```
Array to string
```

```
{
   $reduce: {
     input: ["a", "b", "c"],
     initialValue: "",
     in: { $concat : ["$$value", "$$this"] }
   }
}
```

```
"Abc"
```

```
------------------------------------------------------------
```

```
Sum of array
```

```
{
  $reduce: {
     input: [ 1, 2, 3, 4 ],
     initialValue: { sum: 5, product: 2 },
     in: {
        sum: { $add : ["$$value.sum", "$$this"] },
        product: { $multiply: [ "$$value.product", "$$this" ] }
     }
  }
}
```

```
{ "sum" : 15, "product" : 48 }
```

```
----------------------------------------------------------
```

Multiple search | optional search | optional find | multiple find | $in operator |

```
accountledger.find({
    'accountledger': { $in: [
        ledgers.purchase,
        ledgers.discount,
        ledgers.lessDiscount,
        ledgers.roundOff,
        // ledgers.roundOffAdd,
        // ledgers.roundOffMinus
    ]},
    isdeleted: 0
})
```

—--

Subarray | sub array | subarray db design | sub array db design | array only container objectId

```
    categories: {
        type: [{ type : ObjectId, required: true }],
    },
```

—---

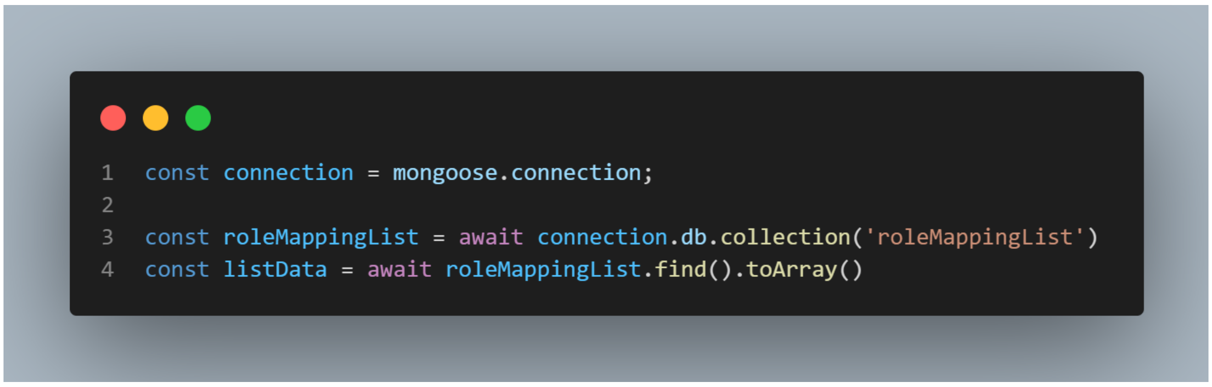
MongoDB Connection || access db in mongoose || access collection in mongoose || access db collection in mongoose || get db in controller ||
get collections in controller || collection not in model
[note: this is used to get collections data, that collections not mentioned in models schema]

```
// ? mongo db connection start
const connection = mongoose.connection;
// ? mongo db connection end


  testView(req, res) {
    connection.db.collection("transactions", function(err, collection){
      collection.find({}).toArray(function(err, data){
        console.log(data); // data printed in console
      })
    });
```

}

// --------------- or --------------------- //



```
const connection = mongoose.connection;

const roleMappingList = await connection.db.collection('roleMappingList')
const listData = await roleMappingList.find().toArray()
```

```
const connection = mongoose.connection;

const getRoleMappingList = async (req, res) => {
    const roleMappingList = await
connection.db.collection('roleMappingList')
    const listData = await roleMappingList.find().toArray()
}
```

// --------------- or --------------------- //

```
var mongoose = require("mongoose");
mongoose.connect(' database_url ');
var conn = mongoose.connection;
conn.on('error', console.error.bind(console, 'connection error:'));
conn.once('open', function () {

conn.db.collection(" collection ", function(err, collection){

   collection.find({}).toArray(function(err, data){
      console.log(data); // data printed in console
   })
});
```

// check connection
mongoose.connection.on("open", function(){ console.log("mongodb is connected!!"); });

// —---------------------or —---------------------------- //

```
const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error: "));
db.once("open", function () {
  console.log("Connected successfully");
});
```

—---

MongoDB Function:
=================

collections.count({isdeleted: 0}).then(data => log(data)).catch(err => log(err))   // return number of document.

collections.find({isdeleted: 0}).sort(_id: -1).then(data => log(data)).catch(err => log(err))   // sort by _id.

collections.findOne({_id: xxxx}) // is faster than find // use findOne for Edit and Update Operations

Index:
 - in case of your collection have 10 million records. then search record using "find({student_id: 1000})" it will take some time.
 - to solve this performance issue use index.
 - create index of unique fields for this record "student_id" is unique
        db.students.ensureIndex({"student_id": 1})
 - then search frecord using find "find({"student_id": 100000})" fraction of secord you will be get result data because we impleted index.
 - be careful using index because index using unique values only.
 - remove index using
        db.students.dropIndex({"student_id": 1})

Study:

version coding structure
Populate [next level of $lookups]

Match 2 objectId in array prototype | match two object Id in array prototype | Match 2 objectId in array method| match two object Id in array method

User _id.equals(another_id)

```
userDetails.find(({_id}) => _id.equals(userId))
```

Duplicate objectId id array | remove duplicate from array

Notes: before remove duplicate objectid need to convert all object as string

```
attributes = [
                "61aa04c7d01002376896ca5a",
                "61ed4d6e94da5baa1c2f19d2",
                "61ed4d3894da5baa1c2f19cf",
                "61ed4d2094da5baa1c2f19cc",
                "61ed4f7894da5baa1c2f19df",
                "61aa04c7d01002376896ca5a",
                "61ab2f9dcb7a0c1cdc0b8109",
                "61ed4d6e94da5baa1c2f19d2",
                "61ed4e1394da5baa1c2f19d6",
                "61aa04c7d01002376896ca5a",
                "61ed4d2094da5baa1c2f19cc",
                "61ed4d3894da5baa1c2f19cf",
                "61ed4d6e94da5baa1c2f19d2",
                "61ed4f0f94da5baa1c2f19dc"
            ]

attributes.map(e => e.toString())
removeDuplicate = [...new Set(attributes)]

Output:
[
                "61aa04c7d01002376896ca5a",
                "61ed4d6e94da5baa1c2f19d2",
                "61ed4d3894da5baa1c2f19cf",
                "61ed4d2094da5baa1c2f19cc",
                "61ed4f7894da5baa1c2f19df",
                "61ab2f9dcb7a0c1cdc0b8109",
                "61ed4e1394da5baa1c2f19d6",
                "61ed4f0f94da5baa1c2f19dc"
            ]
```

Create view:

Syntax:

```
MyModel.connection.db.createCollection('myViewName', {
  viewOn: 'existingCollection',
  pipeline: [/* aggregation pipeline here */]
});
```

Code:

```
const connection = mongoose.connection;
```

```
        connection.db.createCollection(
            "itemDropdown",
            {
                "viewOn": "items",
                "pipeline": [
                    {
                        $match: {
                            isdeleted: 0,
                        }
                    },
                ],
                "allowDiskUse": true,
                "collation": {
                    "locale": "simple",
                  }
            }
        ).then((result) => {
            return res.status(200).json({
                status: true,
                message: "Record Found...",
            })
        }).catch((err) => {
            return res.status(200).json({
                 status: false,
                 message: err.message,
             })
         })
```

Notes:
- If you want send the result to the response need to JSON.stringify otherwise get error response.

Populate:

Notes:
- It's an alternate way of $lookup.

Link: https://mongoosejs.com/docs/populate.html

● Use virtual populate.
● Using a populate nested level is possible.

- **When using function in populate using**
  mongoose-deep-populate
    - Package: `npm i mongoose-deep-populate`
      Link:
      https://www.npmjs.com/package/mongoose-deep-populate

Populate:

```
const storySchema = Schema({
  author: { type: Schema.Types.ObjectId, ref: 'Person' },
  title: String,
  fans: [{ type: Schema.Types.ObjectId, ref: 'Person' }]
});

const Story = mongoose.model('Story', storySchema);

Story.

  find().

  populate({

    path: 'fans',

    match: { age: { $gte: 21 } },

    // Explicitly exclude `_id`, see http://bit.ly/2aEfTdB

    select: 'name -_id'

  }).

  exec();
```

Virtual populate:

```
const PersonSchema = new Schema({

  name: String,

  band: String

});


const BandSchema = new Schema({

  name: String

});
```

```
BandSchema.virtual('numMembers', {
  ref: 'Person', // The model to use
  localField: 'name', // Find people where `localField`
  foreignField: 'band', // is equal to `foreignField`
  count: true // And only get the number of docs
});

// Later
const doc = await Band.findOne({ name: 'Motley Crue' }).
  populate('numMembers');
doc.numMembers; // 2
```

Multi level populate:

```
    .populate({
        path: 'attributesDetails',
        match: { isdeleted: 0 },
        select: 'attributes',
        populate: {
            path: 'attributes',
            match: { isdeleted: 0 }
        }
    })
```

Field Encryption:

npm install mongoose-field-encrypt --save-exact

Security Notes

- *Always store your keys and secrets outside of version control and separate from your database.* An environment variable on your application server works well for this.
- Additionally, store your encryption key offline somewhere safe. If you lose it, there is no way to retrieve your encrypted data.
- Encrypting passwords is no substitute for appropriately hashing them. `bcrypt` is one great option. You can also encrypt the password afer hashing it although it is not necessary.
- If an attacker gains access to your application server, they likely have access to both the database and the key. At that point, neither encryption nor authentication do you any good.

Basic

For example, given a schema as follows:

```
const mongoose = require("mongoose");
const mongooseFieldEncryption =
require("mongoose-field-encrypt").fieldEncryption;
const Schema = mongoose.Schema;

const PostSchema = new Schema({
  title: String,
  message: String,
  references: {
    author: String,
    date: Date
  }
});

PostSchema.plugin(mongooseFieldEncryption, { fields: ["message",
"references"], secret: "Default secret key" });

const Post = mongoose.model("Post", PostSchema);

const post = new Post({ title: "some text", message: "hello all" });

post.save({__secret__: "Dynamic secret key"},function(err) {
  console.log(post.title); // some text (only the message field was set to
be encrypted via options)
```

```
  console.log(post.message); // 
a9ad74603a91a2e97a803a367ab4e04d:93c64bf4c279d282deeaf738fabebe89
  console.log(post.__enc_message); // true
});
```

The resulting documents will have the following format:

```
{
  _id: ObjectId,
  title: String,
  message: String, // encrypted salt and hex value as string, e.g.
9d6a0ca4ac2c80fc84df0a06de36b548:cee57185fed78c055ed31ca6a8be9bf20d30328320
0a280d0f4fc8a92902e0c1
  __enc_message: true, // boolean marking if the field is encrypted or not
  references: undefined, // encrypted object set to undefined
  __enc_references: true, // boolean marking if the field is encrypted or
not
  __enc_references_d: String // encrypted salt and hex object value as
string, e.g.
6df2171f25fd1d32adc4a4059f867a82:5909152856cf9cdb7dc32c6af321c8fe69390c359c
6b19d967eaa6e7a0a97216
```

`find` works transparently and you can make new documents as normal, but you should not use the `lean` option on a find if you want the fields of the document to be decrypted. `findOne`, `findById` and `save` also all work as normal. `update` works *only for string fields* and you would also need to manually set the `__enc_` field value to false if you're updating an encrypted field.

From the mongoose package documentation: *Note that findAndUpdate/Remove do not execute any hooks or validation before making the change in the database. If you need hooks and validation, first query for the document and then save it.*

```
Post.findOneAndUpdate({ _id: postId }, { $set: { message: "snoop",
__enc_message: false } }, {__secret__: "Dynamic secret key"});
```

The above also works for non-string fields. See changelog for more details.

Also note that if you manually set the value `__enc_` prefix field to true then the encryption is not run on the corresponding field and this may result in the plain value being stored in the db.

Search over encrypted fields

Note that in order to use this option a *fixed* salt generator must be provided. See example as follows:

```
const messageSchema = new Schema({
  title: String,
  message: String,
  name: String
});

messageSchema.plugin(mongooseFieldEncryption, {
  fields: ["message", "name"],
  secret: "some secret key",
  saltGenerator: function(secret) {
    return "1234567890123456"; // should ideally use the secret to return a
string of length 16
  }
});

const title = "some text";
const name = "victor";
const message = "hello all";

const Message = mongoose.model("Message", messageSchema);

const messageToSave = new Message({ title, message, name });
await messageToSave.save();

// note that we are only providing the field we would like to search with
const messageToSearchWith = new Message({ name });
messageToSearchWith.encryptFieldsSync({__secret__: "Dynamic secret key"});

// `messageToSearchWith.name` contains the encrypted string text
const results = await Message.find({ name: messageToSearchWith.name },
{__secret__: "Dynamic secret key"});

// results is an array of length 1 (assuming that there is only 1 message
with the name "victor" in the collection)
// and the message in the results array corresponds to the one saved
previously
```

Options

- `fields` (required): an array list of the required fields
- `secret` (required): a string cipher which is used to encrypt the data (don't lose this!)
- `useAes256Ctr` (optional, default `false`): a boolean indicating whether the older `aes-256-ctr` algorithm should be used. Note that this is strictly a backwards compatibility feature and for new installations it is recommended to leave this at default.
- `saltGenerator` (optional, default `const defaultSaltGenerator = secret => crypto.randomBytes(16);`): a function that should return either a `utf-8` encoded string that is 16 characters in length or a `Buffer` of length 16.

This function is also passed the secret as shown in the default function example.

Static methods

For performance reasons, once the document has been encrypted, it remains so. The following methods are thus added to the schema:

- `encryptFieldsSync()`: synchronous call that encrypts all fields as given by the plugin options
- `decryptFieldsSync()`: synchronous call that decrypts encrypted fields as given by the plugin options
- `stripEncryptionFieldMarkers()`: synchronous call that removes the encryption field markers (useful for returning documents without letting the user know that something was encrypted)

Multiple calls to the above methods have no effect, i.e. once a field is encrypted and the `__enc_` marker field value is set to true then the ecrypt operation is ignored. Same for the decrypt operation. Of course if the field markers have been removed via the `stripEncryptionFieldMarkers()` call, then the encryption will be executed if invoked.

Searching

To enable searching over the encrypted fields the `encrypt` and `decrypt` methods have also been exposed.

```
const fieldEncryption = require('mongoose-field-encrypt')
const encrypted = fieldEncryption.encrypt('some text', 'secret'));
const decrypted = fieldEncryption.decrypt(encrypted, 'secret')); //
decrypted = 'some text'
```

Testing

1. Install dependencies with `npm install` and install mongo if you don't have it yet.
2. Start mongo with `mongod`.
3. Run tests with `npm test`. Additionally you can pass your own mongodb uri as an environment variable if you would like to test against your own database, for e.g.

```
URI='mongodb://username:password@127.0.0.1:27017/mongoose-field-encry
pt-test' npm test
```

Pagination:

```
// get page limit
const getPageLimit = (limit) => (parseInt(limit) > 0 ? parseInt(limit)
: 10);

// get page no
const getPageNo = (pageNo) => (parseInt(pageNo) > 0 ? parseInt(pageNo)
: 1);

// get pagination skip value
const getSkip = (limit, pageNo) => limit * (pageNo - 1);

const getPaginationValues = (query) => {
    let { pageNo, limit } = query;
    limit = getPageLimit(limit);
    pageNo = getPageNo(pageNo);
    const skip = getSkip(limit, pageNo);
    return { limit, pageNo, skip };
};

const samplePurchaseList = async ({ query = {} }) => {
    const { skip, limit, pageNo } = getPaginationValues(query);

    let purchases = await transaction.find({ isdeleted: 0 })
        .skip(skip)
        .limit(limit)
        .lean();

   const transactions= await transaction.find({ isdeleted: 0 });
    return {
        statusCode: 200, data: {
            totalCount: transactions.length,
            totalPages: Math.ceil(transactions.length / limit),
            currentPage: pageNo,
            purchases,
        }
    }
}
```

——

——
file upload | multiple file upload

Angular:

HTML:
```
<!-- ^ Attachement -->
<div class="col-sm-12 col-lg-12 col-md-12 col-xl-12">
   <div class="form-group mb-3 file-upload ">
       <span><span textHover class="m-3">Attachement</span> <button
type="button" ngbTooltip="Attachement" class="btn btn-sm"><i
*ngIf="!f._id.value" textHover class='bx bx-upload file-upload-icon
attachement' (click)="fileUpload.click()"></i></button></span>
       <input type="file" multiple hidden #fileUpload
(change)="fileAttachementUpload($event)" id="attachement"
[readonly]="f._id.value" />
       <div *ngIf="f.attachment.value" class="mt-3 ml-3">
  <span *ngFor="let item of f.attachment.value;let i = index;">
      <i primarycolor class='bx bxs-file-blank mr-2'></i>
      {{item.attachment_name}}
      <a [href]="baseURL+item.attachment_path" target="_blank"><i textHover
class='bx bx-link-external ml-5' style="cursor: pointer"></i></a>
      <i *ngIf="!f._id.value" (click)="removeFile(i)" class='bx
bxs-message-square-x ml-5 required-star' style="cursor: pointer"></i><br>
  </span>
</div>

   </div>
</div>
```

TS:
```
// ? file upload start
fileChangeEvent(fileInput: any) {
  const filesToUpload: Array<File> = <Array<File>>fileInput.target.files;
  return filesToUpload
}
fileAttachementUpload(event) {
  const formData: any = new FormData();
  const files: Array<File> = this.fileChangeEvent(event);
  if(files && files.length) {
    for(let i =0; i < files.length; i++){
      formData.append("attachment[]", files[i], files[i]['name']);
    }
    this.transactionService.paymentFileUpload(formData).subscribe(res => {
      const {status, message, data} = res
      if(status) {
        const {attachment} = this.commonform.value
        const updatedAttachment = [...attachment, ...data]
        this.commonform.patchValue({
          attachment: updatedAttachment
        })
      } else {
        this.openSnackBar('file not uploaded')
      }
    })
  }
}
// ? file upload end
```

NodeJS:

```
const multer = require("multer");

// * file upload
const attachmentStorageAndFileName = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, './public/billing/transactions/payment')
    },
    filename: function(req, file, cb) {
        const { originalname, mimetype } = file
        cb(null, `${Date.now()}_file_name-${originalname}`)
    }
})
const attachment = multer({ storage: attachmentStorageAndFileName })
router.post('/attachment', attachment.array("attachment[]"), async (req, res)
=> {
    const { files } = req
    const data = await files.map(({path: attachment_path, originalname:
attachment_name, size: attachment_size, mimetype}) => {
        const [ ,attachment_type] = mimetype.split("/");
        attachment_path = attachment_path.replace('public', '')
        return {attachment_path, attachment_name, attachment_size,
attachment_type }
    })
    return await response(res, 200, true, 'File Uploaded Successfully', data)
})


// * file response
// {
//     fieldname: 'myFile',
//     originalname: 'Final Resume.pdf',
//     encoding: '7bit',
//     mimetype: 'application/pdf',
//     destination: 'public/files',
//     filename: '54a87baf681a51490eda5626f495df6c',
//     path: 'public\\files\\54a87baf681a51490eda5626f495df6c',
//     size: 2034370
// }

// * file upload
```

```
const attachmentStorageAndFileName = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, './public/billing/transactions/payment')
    },
    filename: function(req, file, cb) {
        const { originalname, mimetype } = file
        cb(null, `${Date.now()}_file_name-${originalname}`)
    }
})
const attachment = multer({ storage: attachmentStorageAndFileName })
router.post('/attachment', attachment.array("attachment[]"), async (req, res)
=> {
    const { files } = req
    const data = await files.map(({path: attachment_path, originalname:
attachment_name, size: attachment_size, mimetype}) => {
        const [ ,attachment_type] = mimetype.split("/");
        return {attachment_path, attachment_name, attachment_size,
attachment_type }
    })
    return await response(res, 200, true, 'File Uploaded Successfully', data)
})


// * file response
// {
//      fieldname: 'myFile',
//      originalname: 'Final Resume.pdf',
//      encoding: '7bit',
//      mimetype: 'application/pdf',
//      destination: 'public/files',
//      filename: '54a87baf681a51490eda5626f495df6c',
//      path: 'public\\files\\54a87baf681a51490eda5626f495df6c',
//      size: 2034370
// }
```

—--

Server side search | table search | search value in table

```
collections.find({
      {
            $or: [
                  { name: { $regex: ‘key’, $options: ‘i’ } },
                  { age: { $regex: ‘key’, $options: ‘i’ } }
            ]
      }
})
```

—--

JWT Token:

Token.js

```
const jwt = require("jsonwebtoken");
const { response } =
require("../../functions/response/successResponse");
const { log } = require("../../functions/small-functions/functions");
var env = require('dotenv').config()

const tokenGenerator = (data) => {
    const token = jwt.sign(
        data,
        process.env.JWTSECKEY,
        {expiresIn: "5hours"}
        )
    return token;
}
const tokenValidator = (token) => {
    try {
        const data = jwt.verify(
            token,
            process.env.JWTSECKEY
            )
        return data
    } catch(err) {
        return false
    }
}
const tokenVerify = async (req, res, next) => {
    const {jwtToken} = req.cookies;
    const valid = await tokenValidator(jwtToken)
    if(valid) next(); else response(res, 200, false, 'Access Denied')
}

module.exports = {
    tokenGenerator,
    tokenVerify,
};
```

After login

```
                      // * jwt
            const token = await tokenGenerator({name, mail, mobile,
picture})
```

```
            res.cookie("jwtToken", token)

```

Routes

```

// * protected
router.get('/protected', tokenVerify, (req, res) => res.send('I am
protected'))
```

MONGODB →
Subarray lookup →

```
product.aggregate([{
                $match: {
                    isdeleted: 0,
                                                            _id:   new
mongoose.Types.ObjectId(req.body._id)
                }
            },
            {
                $lookup: {
                    from: "categories",
                    localField: "categoryname",
                    foreignField: "_id",
                    as: "categorydetails"
                }
            },
            {
                $unwind: "$categorydetails"
            },
            {
                $lookup: {
                    from: "subcategories",
                    localField: "subcategoryname",
                    foreignField: "_id",
                    as: "subcategorydetails"
                }
            },
            {
                $unwind: "$subcategorydetails"
            },
            {
                $lookup: {
```

```
                from: "units",
                localField: "totalproductweightunitdd",
                foreignField: "_id",
                as: "totalproductweightunitdddetails"
            }
        },
        {
            $unwind: "$totalproductweightunitdddetails"
        },

        {


            $lookup: {
                from: "units",
                localField: "metaltypearr.weightunitdd",
                foreignField: "_id",
                as: "weightunitdddetails"
            }
        },
        {
            $unwind: "$weightunitdddetails"
        },

        {
            $lookup: {
                from: "attributes",
                localField: "metaltypearr.metaltype",
                foreignField: "_id",
                as: "metaltypedetails"
            }
        },
        {
            $unwind: "$metaltypedetails"
        },
        {
            $unwind: "$metaltypearr"
        },
        {
            $group: {
                _id: {
                    _id: "$_id",
                    productcode: "$productcode",
                    productname: "$productname",
                                                    "category":
"$categorydetails.categoryname",
                                                 "subcategory":
"$subcategorydetails.subcategoryname",
                        categoryname: "$categoryname",
                             "totalproductweightunitdddetails":
"$totalproductweightunitdddetails.unitname",
                        subcategoryname: "$subcategoryname",
                                            totalproductweight:
"$totalproductweight",
                                      totalproductweightunitdd:
"$totalproductweightunitdd",
                        productimage: "$productimage",
                        totalamount: "$totalamount",
                        makingcahrge: "$makingcahrge",
                              makingcahrgediscountpercenttage:
"$makingcahrgediscountpercenttage",
                                            wastagepercenttage:
"$wastagepercenttage",
                                    wastagediscountpercenttage:
"$wastagediscountpercenttage",
                        overalldiscount: "$overalldiscount",
                        sliderimage: "$sliderimage",
                        gstpercentage: "$gstpercentage",
                        aboutproduct: "$aboutproduct",
                        // metaltypearr: "$metaltypearr",
                        video: "$video",
                        notes: "$notes",
                        makingtime: "$makingtime",
                    },
                    metailtypes: {
                        $push: {
                                                   "metaltype":
"$metaltypearr.metaltype",
                                               "metaltypename":
"$metaltypedetails.metaltype",
                            "weight": "$metaltypearr.weight",
                                                "weightunitdd":
"$metaltypearr.weightunitdd",
                                            "weightunitddname":
"$weightunitdddetails.unitname",
                                                "priceperunit":
"$metaltypearr.priceperunit",
                            "amount": "$metaltypearr.amount",
                        }
                    }
                }
            },
            {
                $project: {
                    _id: "$_id._id",
                    productcode: "$_id.productcode",
                    productname: "$_id.productname",
                    "category": "$_id.category",
                    "subcategory": "$_id.category",
                    categoryname: "$_id.categoryname",
                             "totalproductweightunitdddetails":
"$_id.totalproductweightunitdddetails",
                    subcategoryname: "$_id.subcategoryname",
                                            totalproductweight:
"$_id.totalproductweight",
                                      totalproductweightunitdd:
"$_id.totalproductweightunitdd",
                    productimage: "$_id.productimage",
                    totalamount: "$_id.totalamount",
                    makingcahrge: "$_id.makingcahrge",
                              makingcahrgediscountpercenttage:
"$_id.makingcahrgediscountpercenttage",
                                            wastagepercenttage:
"$_id.wastagepercenttage",
                                    wastagediscountpercenttage:
"$_id.wastagediscountpercenttage",
                    overalldiscount: "$_id.overalldiscount",
                    sliderimage: "$_id.sliderimage",
                    gstpercentage: "$_id.gstpercentage",
                    aboutproduct: "$_id.aboutproduct",
                    // metaltypearr: "$_id.metaltypearr",
                    video: "$_id.video",
                    notes: "$_id.notes",
                    makingtime: "$_id.makingtime",
                    metailtypes: "$metailtypes",
                }
            },
            {
                $unwind: "$sliderimage"
            },
            {
                $match: {
```

```
                    "sliderimage.isdeleted": 0,
                }
            },
            {
                $group: {
                    _id: {
                        _id: "$_id",
                        productcode: "$productcode",
                        productname: "$productname",
                                                        "category":
"$categorydetails.categoryname",
                                                     "subcategory":
"$subcategorydetails.subcategoryname",
                        categoryname: "$categoryname",
                            "totalproductweightunitdddetails":
"$totalproductweightunitdddetails.unitname",
                        subcategoryname: "$subcategoryname",
                                             totalproductweight:
"$totalproductweight",
                                       totalproductweightunitdd:
"$totalproductweightunitdd",
                        productimage: "$productimage",
                        totalamount: "$totalamount",
                        makingcahrge: "$makingcahrge",
                              makingcahrgediscountpercenttage:
"$makingcahrgediscountpercenttage",
                                             wastagepercenttage:
"$wastagepercenttage",
                                     wastagediscountpercenttage:
"$wastagediscountpercenttage",
                        overalldiscount: "$overalldiscount",
                        // sliderimage: "$sliderimage",
                        gstpercentage: "$gstpercentage",
                        aboutproduct: "$aboutproduct",
                        video: "$video",
                        notes: "$notes",
                        makingtime: "$makingtime",
                        metailtypes: "$metailtypes"
                    },
                    sliderimages: {
                        $push: {
                            "_id": "$sliderimage._id",
                                                       "imagepath":
"$sliderimage.imagepath",
                                                   "isdeleted":
"$sliderimage.isdeleted",
                        }
                    }

                }
            },
            {

                $project: {
                    "_id": "$_id.productcode",
                    _id: "$_id._id",
                    productcode: "$_id.productcode",
                    productname: "$_id.productname",
                    "category": "$_id.category",
                    "subcategory": "$_id.category",
                    categoryname: "$_id.categoryname",
                            "totalproductweightunitdddetails":
"$_id.totalproductweightunitdddetails",
                    subcategoryname: "$_id.subcategoryname",
                                            totalproductweight:
"$_id.totalproductweight",
                                      totalproductweightunitdd:
"$_id.totalproductweightunitdd",
                    productimage: "$_id.productimage",
                    totalamount: "$_id.totalamount",
                    makingcahrge: "$_id.makingcahrge",
                              makingcahrgediscountpercenttage:
"$_id.makingcahrgediscountpercenttage",
                                            wastagepercenttage:
"$_id.wastagepercenttage",
                                    wastagediscountpercenttage:
"$_id.wastagediscountpercenttage",
                    overalldiscount: "$_id.overalldiscount",
                    gstpercentage: "$_id.gstpercentage",
                    aboutproduct: "$_id.aboutproduct",
                    video: "$_id.video",
                    notes: "$_id.notes",
                    makingtime: "$_id.makingtime",
                    metailtypes: "$_id.metailtypes",
                    sliderimage: "$sliderimages",
                }

            }
```

```
    ]).
```

Pipeline Lookup : →

```
        {
            $lookup: {
                from: "executives",
                "let": { "salesmangerid": "$_id" },
                "pipeline": [{
                    "$match": {
                        isdeleted: 0,
                                                "createdAt": { $gte: new Date(`${req.body.year}-${req.body.month}-01T00:00:00.000+00:00`), $lt: new Date(`${req.body.year}-${req.body.month}-31T00:00:00.000+00:00`), },
                        "$expr": {
                            $and: [
                                { $eq: ["$salesmanager", "$$salesmangerid"] },


                            ]
                        },
                    },
                },
                // {
```

```
                //    $group: {
                //        _id: {
                //            _id: null
                //        },
                //        total: {
                //            $sum: "$amount"
                //        }
                //    },

                // },
                {
                    $project: {
                        _id: "$_id",
                        target: "$target",
                        name: "$name",
                    }
                }
            ],
            as: "salesexecutivelist"
        }
    },
    {
```

```
                $unwind: "$salesexecutivelist"

            },
```


MongoDB | Lookup Empty Array | preserveNullAndEmptyArrays | unwind | when lookup using empty array field | when lookup using empty fields

|-> lookup using optional fields then the result of the lookup array only show values in lookup array, when using preserveNullAndEmptyArrays empty fields viewed in lookup results

```
{ $lookup: { from: "nation", localField: "nation", foreignField: "_id", as: "z" } },

{ $unwind: { path: "$x", "preserveNullAndEmptyArrays": true } },
```

MongoDB DateString →

```
                    $dateToString: {
                        format: "%d/%m/%Y",
                        date: "$validfrom",
                    },
```

MongoDB Array to string | array of object to string | reduce | $reduce

```
db.collection.aggregate([

  {

      $project: {

          field_value: {

              $reduce: {

                  input: "$field_value",

                  initialValue: "",

                  in: {
```

```
                  $concat: [
                     "$$value",
                     { $cond: [ { $eq: [ "$$value", "" ] }, "", "," ] },
                     { $toString: "$$this.name" }
                  ]
               }
            }
         }
      }
   }
])
```

MONGOOSE Object iD

```
$match: {
state: new mongoose.Types.ObjectId(req.body.state)
}
```

OR Condition | Match with or condition | or condition in match | condition based match | or condition mongo db | match or condition

```
            {
               $match: {
                  $or: [
                     {username: req.body.username},
                     {mobile:Number(req.body.username)},
              ],
             isactive: true,
             isdeleted: 0,
             password: req.body.password,
           }
         },
```

MONGODB:

```
daysCount: {
              $round: { $divide: [{ $subtract: ["$to", "$from"] },
86400000] }
       }
```

Directories | folder | make folder | make directory

```
const fs = require('fs');
const moment = require('moment');

const checkIfFolderIsExist = (dir) => fs.existsSync(dir)
const makeFolder = (path, dir) => {
    const folderName = `./public/futureVision/version1/${path}/${dir}`
    if (!checkIfFolderIsExist(folderName)) fs.mkdirSync(folderName, {
recursive: true });
}
const renameFolder = (path, oldName, newName) => {
    const oldFolderName =
`./public/futureVision/version1/${path}/${oldName}`
    const newFolderName =
`./public/futureVision/version1/${path}/${newName}`
    if (oldFolderName !== newFolderName &&
checkIfFolderIsExist(oldFolderName)) {
        fs.rename(oldFolderName, newFolderName, function(err) {
            if (err) console.log(err); else console.log("Successfully 
renamed the folder.")
        })
    } else {
        makeFolder(path, newName)
    }
}
const deleteFolder = (path, dir) => renameFolder(path, dir, 
`${dir}-deletedAt-${moment().valueOf()}`)

module.exports = {
    makeFolder,
    renameFolder,
    deleteFolder,
    checkIfFolderIsExist,
};
```

Express-validator

Custom | validation | validation in routes

Condition based validation

```
const parallelValidate = validations => {
    return async (req, res, next) => {
      await Promise.all(validations.map(validation => 
validation.run(req)));

      const errors = validationResult(req);
      if (errors.isEmpty()) {
        return next();
      }

      const erroryHandle = await [...errors.array().map(err => 
err.msg)]
      return errorResponse(res, 200, false, erroryHandle.toString(), 
errors.array() )
    };
  };
```

```

```

```
router.post('/add',  tokenVerify, parallelValidate([
    body('unit', 'unit is
required').notEmpty().isMongoId().withMessage('unit it mongo id'),
    body('type', 'type is requried').notEmpty().isIn(['document',
'video']).withMessage('type must be document or video'),
    // ? documnet validation start
    body('type').custom((value, {req}) => {
        if(value === 'document') {
            const { document } = req.body
            if(document && document.length) {
                return document.every(({name, attachment: { name:
a_name, path, type, size }}) => name && a_name && path && type && size)
            } else return false
        } return true
    }).withMessage('documents is required'),
    // ? documnet validation end
]),
async (req, res) => {
    await add(req).then(({statusCode, status, message, data = null}) =>
        response(res, statusCode, status, message, data)
    )}
)
```

Update subarray | subArray update | push new value array | append new value in array | push new value in array | push value in array | push subarray

```
            const object= {
                studentId,
                "$push": {
                    "payment": payment,
                    "updatedby": getupdatedbyObject(updatedby,
'addNewFeeData')
                },
            }
```

```
            const updateData = await
findData.updateOne(object).catch(err => catchError(err))
```

Update subarray element | update particular property in sub array element | update particular element in subarray | arrayFilters | update

subarray based on condition | find and update subarray | update element in nested array based on condition | update property in nested array based on condtion

```
db.alumni.insertMany( [
   {
      "_id": 1,
      "name": "Christine Franklin",
      "degrees": [
         { "level": "Master" },
         { "level": "Bachelor" }
      ],
  },
   {
      "_id": 2,
      "name": "Reyansh Sengupta",
      "degrees": [ { "level": "Bachelor" } ],
   }
] )
```

query:

```
db.alumni.updateMany(
   { },
   { $set : { "degrees.$[degree].gradcampaign" : 1 } },
   { arrayFilters : [ {"degree.level" : { $ne: "Bachelor" } } ] }
)
```

————————————

```
db.students4.insertOne(
   { "_id" : 1,
      "grades" : [
        { type: "quiz", questions: [ 10, 8, 5 ] },
        { type: "quiz", questions: [ 8, 9, 6 ] },
        { type: "hw", questions: [ 5, 4, 3 ] },
        { type: "exam", questions: [ 25, 10, 23, 0 ] },
      ]
   }
)
```

Query:

```
db.students4.updateMany(
   {},
   { $inc: { "grades.$[t].questions.$[score]": 2 } },
   { arrayFilters: [ { "t.type": "quiz" }, { "score": { $gte: 8 } } ] }
)

.. note::

   The spacing is significant in the array identifier. If you write
   the identifier as ``grades.$[ t ].questions.$[ score ]``, the
   operation will fail.
```

Ex:

```
const updateData = studentFeeSchema.findOneAndUpdate(
    { 'payment._id': paymentId },    // * find condition
    { $set: { 'payment.$[elem].isdeleted': 0 } },    // * updated data
(elem: for subarray conditon in arrayFilters)
    { arrayFilters: [ { 'elem,_id': paymentId } ] }
)
```

```
-----------------------------------------------------------------------
```

Pull | remove element from sub array | delete element from subarray

Data:

```
{
   _id: 1,
   results: [
      { item: "A", score: 5 },
      { item: "B", score: 8, comment: "Strongly agree" }
   ]
}
{
   _id: 2,
   results: [
      { item: "C", score: 8, comment: "Strongly agree" },
      { item: "B", score: 4 }
   ]
}
```

Code:

```
db.survey.update(
  { },
  { $pull: { results: { score: 8 , item: "B" } } },
  { multi: true }
)
----------------------------------------------------------------------
```

isArray | isObject | isString | isEmpty | isBoolean

```
const isArray = arr => arr ? arr.constructor === Array : false
const isObject = obj => obj ? obj.constructor === Object : false
const isString = str => str ? str.constructor === String : false
const isEmpty = value => [null, '', undefined].includes(value) ? true :
false
const isBoolean = val => isEmpty(val) ? false : val.constructor ===
Boolean ? true : false
```

Time | add hours | add time | time conversion | 24 hours to 12 hours

```

const addHours = (hours = loginExpiry?.auth, date = new Date()) => {
    date = new Date(date)
    return new Date(date.setHours(date.getHours() + hours))
}

const sliceNumber = (number = '0', length = 1, concat = '00000') => {
    if(`${number}`.length < length) return (concat +
getNumber(number)).slice(-getNumber(length))
    return number
  }
const timeConvertTo12 = (time) => {
    if (time && time.split(':')?.length && time.split(':')?.length ==
2) {
      const [hours, minutes] = time.split(':')
      if (getNumber(hours) > 12) return
`${sliceNumber(subtractTwoNumber(hours, 12), 2)}:${sliceNumber(minutes,
2)} PM`
      return `${sliceNumber(hours, 2)}:${sliceNumber(minutes, 2)} AM`
    }
    return ''
  }
```

Nodemailer

Package: `npm i nodemailer`

```

const { createTransport } = require('nodemailer');



const transporter = await createTransport(({
    host: "webmail.zerobugz.com",
    port: 465,
    secure: true,
    auth: {
        user: "admin@Zerobugz.com",
        pass: "all4zerobugz123!@#"
    },
}));
const options = {
    from: "zerobugz@gmail.com",
    to: "hariprasanthzerobugz@gmail.com",
    cc: "hariprasanthzerobugz@gmail.com",
    subject: "Nodemailer",
    text: "message"
}
await transporter.sendMail(options, async function (err, info) {
    log('info', info)
    log('err', err)
    if (err) return await errorResponse(res, 200, false, err.message,
err)
    if (info) return await response(res, 200, true, `Mail Send
successfully`)
}
)
```

Scheduler

```
npm install --save node-cron
```

```
var cron = require('node-cron');
```

```
cron.schedule('* * * * *', () => {
  console.log('running a task every minute');
});
```

```
------
```

```
npm install cron
```

```
var CronJob = require('cron').CronJob;
```

```
var job = new CronJob(
     '* * * * * *',

     function() {

          console.log('You will see this message every
second');

     },

     null,

     true,

     'America/Los_Angeles'

);

// Use this if the 4th param is default value(false)

// job.start()
```

Cron | scheduler

`npm i cron`

```
const { CronJob } = require("cron");
const installmentPendingAlert = new CronJob('0 0 10 * * *', function()
{   // Run on 09:18:00 at every day
    // * operation
});
const runScheduler = () => {
    installmentPendingAlert.start()
}
const stopScheduler = () => {
    installmentPendingAlert.stop()
}
module.exports = {
    runScheduler,
    stopScheduler
};
```

—--

Response types

app.js

```
"use strict";
//  https://expressjs.com/en/4x/api.html#res
const express = require("express");
const app = express();
const port = process.env.port || 3000;
app.set("view engine", "pug");
app.set("views", process.cwd() + "/views");

app.get("/", (req, res) => {
  //handle route: get requests for "/"
  // res.send("<h1>asdf</h1>") //looks at content to figure out type
  // res.end() //no type header set
  // res.json({data:123}) //set type as application/json
  // res.redirect(301, "/other");
  // res.format({
  //   "text/plain": () => {
  //     res.send("Just some words");
  //   },
  //   "text/html": () => {
  //     res.send("<h1>Here be HTML</h1>");
  //   },
  //   "application/json": () => {
  //     res.send({ message: "This is a JSON response" });
  //   },
  //   "text/xml":()=>{
  //       res.send('<?xml version="1.0">');
  //   },
  //   "default": () => {
  //     //any other types I don't have
  //     res.status(406).send("Not Acceptable");
  //   }
  // });
  // res.links({
  //   first: "http://localhost:3000/?page=1",
  //   prev: "http://localhost:3000/?page=2",
  //   next: "http://localhost:3000/?page=4",
  //   last: "http://localhost:3000/?page=9",
  //   canonical: "http://localhost:3000/page/3",
  //   prefetch: "http://localhost:3000/something.png",
  //   preload: "http://localhost:3000/something-else.png"
  // });
  // let locals = { name: "jeffrey" };
  // res.render("myview", locals, (err, html) => {
  //   if (err) {
  //     console.log(err);
  //     return;
  //   }
  //   console.log(html);
  //   res.send(html);
  // });
  // res.set("Content-Type", "text/html") //set any header as the first header
  // res.append("Access-Control-Allow_Origin", "*") //set headers after the first one
  // res.cookie('name', 'Steve', { domain: '.example.com', path: '/',
  //                          secure: true, maxAge: 2592000000}) //30 days
  // res.status(404).end()
  // res.type("application/json") //sets the Content-Type header
  // res.attachment("/path/to/filename.png"); //sets Content-Disposition header
  // res.sendFile("./img/cotton-candy.gif", err => console.log);
  res.download("./img/cotton-candy.gif", "candy.gif", err => {
    console.log(err);
  });
});

app.listen(port, err => {
  if (err) {
    return console.log(err);
  }
  console.log("listening on port", port);
});
```

myView.pug

```
- var email = "dude@abides.com"
doctype html
html
  head
    title sample view page

  body
    p(title='email') #{email}
    p(title='name') #{name}
    p(title='locals') #{settings.views}
```

—---

‘redirect has been blocked by cors policy: no ‘access-control-allow-origin’ header’ ?

app.js

```
var app = express();

app.use((req,res,next)=>{
  res.setHeader('Access-Control-Allow-Origin','*');

res.setHeader('Access-Control-Allow-Methods','GET,POST,PUT,PATCH,DELETE
');

res.setHeader('Access-Control-Allow-Methods','Content-Type','Authorizat
ion');
  next();
})
```

—---

Performance

Package: npm i compression

const compression = require("compression");

var app = express();

// this will compress all responses
app.use(compression())

// or

```
// this will compress all responses
app.use(compression({
  level: 6,
  threshold: 0,
  filter: (req, res) => {
    if(req.headers['authorization']) {
      return false
    }
    return compression.filter(req, res)
  }
}))
```

—---

Time range

```
var range = ['05:00','22:00'];

function isInRange(value, range) {
  return value >= range[0] && value <= range[1];
}

['04:59','08:30','23:15'].forEach(function(time) {
  console.log(time + ' is ' + (isInRange(time, range)? ' ':'not ') + 'in range');
});

// Alternatively
['04:59','23:15','08:30'].forEach(function(time) {
  var inRange = isInRange(time, range);
  console.log(time + ' is in range ' + (inRange? range : range.slice().reverse()).join(' - '));
});
```

—---

ERROR:

typeError [ERR_UNESCAPED_CHARACTERS]: Request path contains unescaped characters

- To encode message to send message in tamil

```
        const encodedMessage = await encodeURIComponent(message)
```

—---