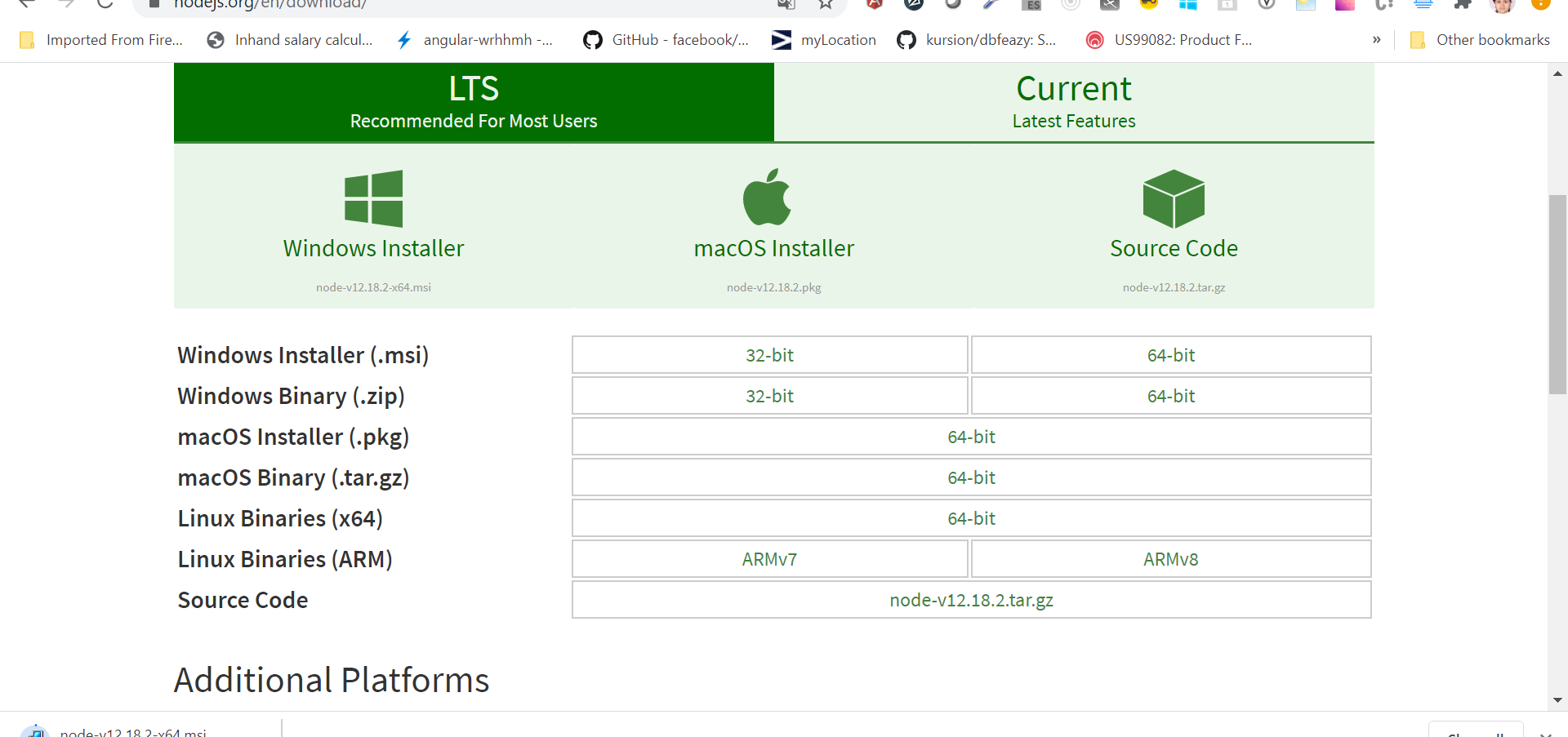
VDN ChatBot Application

**Pre-requisites**

1. **Node.js from url** <https://nodejs.org/en/download/>



1. **Download git from url** <https://git-scm.com/downloads>

**Setup**

1. Clone repository from url : <https://github.com/amruta21/vdn>
2. Add bot id in Index.js where “Telegram\_id” on 2 nd line added . Add it as 1216614278:AAESFgxVOLWNpqkmgqfEzHuxJOPJWSNKJ7Q.
3. Run node index inside vdn folder through commend line

**Telegram Bot Id**: 1216614278:AAESFgxVOLWNpqkmgqfEzHuxJOPJWSNKJ7Q

**Code Walkthrogh :**

Prerequisite Modules with versions:

"date-and-time": "^0.13.1",

"dbf": "^0.1.4",

"dbffile": "^1.4.1",

"dbfstream": "^1.0.24",

"moment": "^2.27.0",

"node-telegram-bot-api": "^0.50.0",

"stream-dbf": "^0.3.0"

Step By Step understanding :

1. Telegram bot initialization

var TelegramBot = require('node-telegram-bot-api'),

    // Be sure to replace YOUR\_BOT\_TOKEN with your actual bot token on this line.

    telegram = new TelegramBot("1216614278:AAESFgxVOLWNpqkmgqfEzHuxJOPJWSNKJ7Q", { polling: true });

we are using telegram id for initialization so that messages will be send through same chatbot.

1. Created Operations file for repeated types of functions

var operations = require("./operations/operations");

We already have operations.js file where we are doing file operations as read , write and update.

1. Operations file Details
2. Modules initialization

var dbf = require('dbf'),

 fs = require('fs');

var DBF = require('stream-dbf');

var moment = require("moment");

required modules initialization

1. Creation of buffer to insert data in table

function toBuffer(ab) {

    var buffer = new Buffer.alloc(ab.byteLength);

    var view = new Uint8Array(ab);

    for (var i = 0; i < buffer.length; ++i) {

        buffer[i] = view[i];

    }

    return buffer;

}

1. Update Operation for SEZGDGK table

const update = function(allListRecords, lastUsedValue){

    allListRecords.forEach(element => {

        if(element.FIELDID == 'PIECERAT'){

                element.LASTUSED = lastUsedValue;

        }

    });

    var buf1 = dbf.structure(allListRecords);

    fs.writeFile('./tables/SEZGDGK.DBF', toBuffer(buf1.buffer), function (err) {

        if (err) throw err;

        console.log('Saved!');

      });

    return true;

}

Here we are checking weather PIECERAT field is current field id then check value from array to total record updated

Example.. If we already have 5 records in QTCNCHITIET then if we are sending msg with 2 correct records then PIECERAT field value we are replacing with 7.

And fs.write will updated all records.

1. Read file data and insert in object against any key

const read = function(path, keyVal, value) {

    let arrayId = {};

    var parser = new DBF(path);

    parser.stream.on('data', function(record) {

        if(value === 'ALLOWMULTI') {

            buf = Buffer.from(record.ALLOWMULTI);

            record.ALLOWMULTI = buf.toJSON().data[0];

        }

        arrayId[record[keyVal]]  = value == "True"? value: record[value];

    });

    return arrayId;

}

Here we are returning object with key whatever we are passing and value against it with parser.stream module.

1. Get Recorded quantity against any jobid

const getRecordedQuantity = function(records, jobId) {

    let recordedQuantity = 0;

    for (let element of records) {

        if(Number(element.STEPCODE.trim()) == jobId) {

            recordedQuantity += element.QUANTITY;

        }

    };

    return recordedQuantity;

}

Here we are just summing up recorded quantity with QTCNCHITIET table against jobId

1. Check weather current user sending message is valid user or not against EPL table

const isValidUser = function(userArray, userId, id, firstName, arrayIdMultiUser) {

    var flag = false;

    console.log(arrayIdMultiUser, userId, firstName);

    if(userArray[userId] === firstName || arrayIdMultiUser[firstName] === 1) {

        flag = true;

    }

    return flag;

}

Check weather multiuser allowed or not or userid matches with current username.

1. To check monthly earning check if date is valid then compute earning

const isValidDate = function(date) {

    var y = date.slice(0, 4);

    var m = date.slice(4, 6);

    var d = date.slice(6, 8);

    var quantityDate = moment(y+'-'+m+'-'+d);

    var currentDate = moment();

    var firstdate = moment().startOf('month');

    if(quantityDate <= currentDate && quantityDate >= firstdate) {

        return true;

    } else {

        return false;

    }

}

Here trying to check current date , 1 st day of month and quantity date according to query .

1. Initialization and read operation from respective table

var DBF = require('stream-dbf');

const arrayId = operations.read("./tables/EPL.DBF", 'EPL\_ID', "True");

const arrayIdName = operations.read("./tables/EPL.DBF", 'EPL\_ID', 'TELUSERNAM');

const arrayIdMultiUser = operations.read("./tables/EPL.DBF", 'TELUSERNAM', 'ALLOWMULTI');

const arrayProductId = operations.read("./tables/gmstyle.DBF", 'STL\_ID', "True");

const arraySMV = operations.read("./tables/gmstyle.DBF", 'EPL\_ID', 'SMV\_RATE');

const arrayRQR = operations.read("./tables/gmstyle.DBF", 'STL\_ID', 'STL\_RQRQTY');

var parserJobID = new DBF("./tables/qtcnchitiet.DBF");

var parserList = new DBF("./tables/SEZGDGK.DBF");

var parserCompany = new DBF("./tables/company.DBF");

var arrayQuantityId = {};

var arrayJobID = {};

var arrayEarning = {};

var arrayHeso = {};

var lastUsedValue, LBCOSTPERS;

var allListRecords = [];

var arraySochitiet = [];

var allListRecordsQuantity = [];

var monthEarning = 0;

var styleEarning = 0;

var DBFFile = require('dbffile');

1. Read record from QTCNCHITIET Table with earning calculations

async function readRecord(arrayQuantityId, userId, prodctID) {

    let dbf =  await DBFFile.DBFFile.open('./tables/PIECERATERECORD.DBF')

    let records =  await dbf.readRecords(100000);

    for (let record of records) {

        if(record.ROWID.includes("bot")) {

            arrayQuantityId[record.STEPCODE+":"+record.ROWID] = record.QUANTITY;

        } else {

            arrayQuantityId[record.STEPCODE] = record.QUANTITY;

        }

        const flagValid = operations.isValidDate(record.DATE);

        if(flagValid === true && record.WORKER.trim().toUpperCase() === userId.toUpperCase() && record.C1.trim().toUpperCase() === prodctID.toUpperCase()) {

            monthEarning += getTotalEarning(record.QUANTITY, arrayEarning[record.STEPCODE.trim().toString()], LBCOSTPERS, arrayHeso[record.STEPCODE.trim().toString()], arraySMV[record.QUANTITY]);

        }

        if(record.WORKER.trim().toUpperCase() === userId.toUpperCase() && record.C1.trim().toUpperCase() === prodctID.toUpperCase()) {

            styleEarning += getTotalEarning(record.QUANTITY, arrayEarning[record.STEPCODE.trim().toString()], LBCOSTPERS, arrayHeso[record.STEPCODE.trim().toString()], arraySMV[record.QUANTITY]);

        }

        allListRecordsQuantity.push(record);

    }

}

Here we are using different library as DBFFile where we added some custom code acccoring to our DBF format and then after reading data from file one by one calculation style earning and monthly earning.

1. To write records in same file we are using DBFFile library only

async function recordWrite(records) {

    let dbf = await DBFFile.DBFFile.open('./tables/PIECERATERECORD.DBF');

    await dbf.appendRecords(records);

    console.log(`${records.length} records added.`);

}

Here we are just appending data to QTCNCHITIET Table.

1. Updated operations from operations file with value passing

async function updateTable(records, lastValue) {

    operations.update(records, lastValue);

}

1. On message from telegram we calculate everything from operations and earning

telegram.on("text", (message) => {

 //////calculations

});

1. Earning calculation

function getTotalEarning(quantity, allowed\_time, LBCOSTPERS, heso, smvRate = 1) {

    return (parseFloat(quantity) \* parseFloat(allowed\_time) \* parseFloat(LBCOSTPERS) \* parseFloat(heso) \* parseFloat(smvRate));

}

Above code is step by step implementation for telegram