**k3Controller.js Documentation**

**Overview**

The k3Controller.js file is a module that handles the logic for the K3 game in a software project. It includes functions for rendering the K3 page, validating and processing bets, generating random numbers, and distributing commissions.

**Functions**

**K3Page**

A function that renders the K3 page.

*const* K3Page = *async* (*req*, *res*) *=>* {

    return *res*.render("bet/k3/k3.ejs");

}

**betK3**

A function that processes a bet for the K3 game.

*const* betK3 = *async* (*req*, *res*) *=>* {

    try {

*let* { listJoin, game, gameJoin, xvalue, money } = *req*.body;

*let* auth = *req*.cookies.auth;

        // let validate = await validateBet(join, list\_join, x, money, game);

        // if (!validate) {

        //     return res.status(200).json({

        //         message: 'Đặt cược không hợp lệ',

        //         status: false

        //     });

        // }

*const* [k3Now] = await connection.query(`SELECT period FROM k3 WHERE status = 0 AND game = ${game} ORDER BY id DESC LIMIT 1 `);

*const* [user] = await connection.query('SELECT `phone`, `code`, `invite`, `level`, `money` FROM users WHERE token = ? AND veri = 1  LIMIT 1 ', [auth]);

        if (k3Now.length < 1 || user.length < 1) {

            return *res*.status(200).json({

                message: 'Error!',

                status: false

            });

        }

*let* userInfo = user[0];

*let* period = k3Now[0];

*let* date = new *Date*();

*let* years = formateT(date.getFullYear());

*let* months = formateT(date.getMonth() + 1);

*let* days = formateT(date.getDate());

*let* id\_product = years + months + days + Math.floor(Math.random() \* 1000000000000000);

*let* total = 0;

        if (gameJoin == 1) {

            total = money \* xvalue \* (*String*(listJoin).split(',').length);

        } else if (gameJoin == 2) {

*let* twoSame = listJoin.split('@')[0]; // Chọn 2 số phù hợp

*let* motDuyNhat = listJoin.split('@')[1]; // Chọn một cặp số duy nhất

            if (twoSame.length > 0) {

                twoSame = twoSame.split(',').length;

            }

*let* lengthArr = 0;

*let* count = 0;

            if (motDuyNhat.length > 0) {

*let* arr = motDuyNhat.split('&');

                for (*let* i = 0; i < arr.length; i++) {

                    motDuyNhat = arr[i].split('|');

                    count = motDuyNhat[1].split(',').length;

                }

                lengthArr = arr.length;

                count = count;

            }

            total = money \* xvalue \* (lengthArr \* count) + (twoSame \* money \* xvalue);

        } else if (gameJoin == 3) {

*let* baDuyNhat = listJoin.split('@')[0]; // Chọn 3 số duy nhất

*let* countBaDuyNhat = 0;

            if (baDuyNhat.length > 0) {

                countBaDuyNhat = baDuyNhat.split(',').length;

            }

*let* threeSame = listJoin.split('@')[1].length; // Chọn 3 số giống nhau

            total = money \* xvalue \* countBaDuyNhat + (threeSame \* money \* xvalue);

        } else if (gameJoin == 4) {

*let* threeNumberUnlike = listJoin.split('@')[0]; // Chọn 3 số duy nhất

*let* twoLienTiep = listJoin.split('@')[1]; // Chọn 3 số liên tiếp

*let* twoNumberUnlike = listJoin.split('@')[2]; // Chọn 3 số duy nhất

*let* threeUn = 0;

            if (threeNumberUnlike.length > 0) {

*let* arr = threeNumberUnlike.split(',').length;

                if (arr <= 4) {

                    threeUn += xvalue \* (money \* arr);

                }

                if (arr == 5) {

                    threeUn += xvalue \* (money \* arr) \* 2;

                }

                if (arr == 6) {

                    threeUn += xvalue \* (money \* 5) \* 4;

                }

            }

*let* twoUn = 0;

            if (twoNumberUnlike.length > 0) {

*let* arr = twoNumberUnlike.split(',').length;

                if (arr <= 3) {

                    twoUn += xvalue \* (money \* arr);

                }

                if (arr == 4) {

                    twoUn += xvalue \* (money \* arr) \* 1.5;

                }

                if (arr == 5) {

                    twoUn += xvalue \* (money \* arr) \* 2;

                }

                if (arr == 6) {

                    twoUn += xvalue \* (money \* arr \* 2.5);

                }

            }

*let* UnlienTiep = 0;

            if (twoLienTiep == 'u') {

                UnlienTiep += xvalue \* money;

            }

            total = threeUn + twoUn + UnlienTiep;

        }

*let* fee = total \* 0.02;

*let* price = total - fee;

*let* typeGame = '';

        if (gameJoin == 1) typeGame = 'total';

        if (gameJoin == 2) typeGame = 'two-same';

        if (gameJoin == 3) typeGame = 'three-same';

        if (gameJoin == 4) typeGame = 'unlike';

*let* checkTime = timerJoin(date.getTime());

*let* check = userInfo.money - total;

        if (check >= 0) {

*let* timeNow = *Date*.now();

*const* sql = `INSERT INTO result\_k3 SET id\_product = ?,phone = ?,code = ?,invite = ?,stage = ?,level = ?,money = ?,price = ?,amount = ?,fee = ?,game = ?,join\_bet = ?, typeGame = ?,bet = ?,status = ?,time = ?,today = ?`;

            await connection.execute(sql, [id\_product, userInfo.phone, userInfo.code, userInfo.invite, period.period, userInfo.level, total, price, xvalue, fee, game, gameJoin, typeGame, listJoin, 0, timeNow, checkTime]);

            await connection.execute('UPDATE `users` SET `money` = `money` - ? WHERE `token` = ? ', [total, auth]);

*const* [users] = await connection.query('SELECT `money`, `level` FROM users WHERE token = ? AND veri = 1  LIMIT 1 ', [auth]);

            await rosesPlus(auth, total);

*const* [level] = await connection.query('SELECT \* FROM level ');

*let* level0 = level[0];

*const* sql2 = `INSERT INTO roses SET phone = ?,code = ?,invite = ?,f1 = ?,f2 = ?,f3 = ?,f4 = ?,time = ?`;

*let* total\_m = total;

*let* f1 = (total\_m / 100) \* level0.f1;

*let* f2 = (total\_m / 100) \* level0.f2;

*let* f3 = (total\_m / 100) \* level0.f3;

*let* f4 = (total\_m / 100) \* level0.f4;

            await connection.execute(sql2, [userInfo.phone, userInfo.code, userInfo.invite, f1, f2, f3, f4, timeNow]);

            return *res*.status(200).json({

                message: 'Successful bet',

                status: true,

                // data: result,

                change: users[0].level,

                money: users[0].money,

            });

        } else {

            return *res*.status(200).json({

                message: 'The amount is not enough',

                status: false

            });

        }

    } catch (error) {

    }

}

The provided code is an async function betK3 which handles placing bets in a gaming application.

1. It extracts required data from the request body and user cookies, performs database queries to retrieve necessary information, calculates the total amount based on the selected game type and input values, deducts the bet amount from user's balance, and updates the database accordingly.
2. It also handles error scenarios where the required data is not found or the user's balance is insufficient for the bet amount. The code structure includes try-catch blocks to handle exceptions that may occur during the process.
3. However, the code has references to external functions like formateT(), connection.query(), and rosesPlus(), which are not defined in the provided snippet.

**addK3**

A function that adds a new K3 game result.

*const* addK3 = *async* (*game*) *=>* {

    try {

*let* join = '';

        if (*game* == 1) join = 'k3d';

        if (*game* == 3) join = 'k3d3';

        if (*game* == 5) join = 'k3d5';

        if (*game* == 10) join = 'k3d10';

*let* result2 = makeid(3);

*let* timeNow = *Date*.now();

*let* [k5D] = await connection.query(`SELECT period FROM k3 WHERE status = 0 AND game = ${*game*} ORDER BY id DESC LIMIT 1 `);

*const* [setting] = await connection.query('SELECT \* FROM `admin` ');

*let* period = k5D[0].period;

*let* gameRepresentationId = GameRepresentationIds.K3[*game*];

*let* NewGamePeriod = generatePeriod(gameRepresentationId);

*let* nextResult = '';

        if (*game* == 1) nextResult = setting[0].k3d;

        if (*game* == 3) nextResult = setting[0].k3d3;

        if (*game* == 5) nextResult = setting[0].k3d5;

        if (*game* == 10) nextResult = setting[0].k3d10;

*let* newArr = '';

        if (nextResult == '-1') {

            await connection.execute(`UPDATE k3 SET result = ?,status = ? WHERE period = ? AND game = "${*game*}"`, [result2, 1, period]);

            newArr = '-1';

        } else {

*let* result = '';

*let* arr = nextResult.split('|');

*let* check = arr.length;

            if (check == 1) {

                newArr = '-1';

            } else {

                for (*let* i = 1; i < arr.length; i++) {

                    newArr += arr[i] + '|';

                }

                newArr = newArr.slice(0, -1);

            }

            result = arr[0];

            await connection.execute(`UPDATE k3 SET result = ?,status = ? WHERE period = ? AND game = ${*game*}`, [result, 1, period]);

        }

*const* sql = `INSERT INTO k3 SET period = ?, result = ?, game = ?, status = ?, time = ?`;

        await connection.execute(sql, [NewGamePeriod, 0, *game*, 0, timeNow]);

        if (*game* == 1) join = 'k3d';

        if (*game* == 3) join = 'k3d3';

        if (*game* == 5) join = 'k3d5';

        if (*game* == 10) join = 'k3d10';

        await connection.execute(`UPDATE admin SET ${join} = ?`, [newArr]);

    } catch (error) {

        if (error) {

        }

    }

}

The given code defines an asynchronous function addK3 that takes a game parameter. Inside the function:

1. A series of conditional statements determine the value of join based on the input game.
2. Various database queries are made using connection.query and connection.execute to retrieve and update data.
3. Values for result2, timeNow, period, gameRepresentationId, and NewGamePeriod are calculated.
4. Further conditional statements update the data and insert new records based on the value of game.
5. Finally, an update query is executed on the admin table based on the value of join and newArr.
6. An error handling block is present but currently empty. This code snippet is dependent on external functions or objects like makeid, GameRepresentationIds, and generatePeriod. It is assumed that these are defined elsewhere in the codebase.

**handlingK3**

A function that processes the K3 game results.

*const* handlingK3 = *async* (*typeid*) *=>* {

*let* game = *Number*(*typeid*);

    await funHanding(game);

    await plusMoney(game);

}

There is a syntax error in the provided code. The arrow function syntax is used which might not be compatible with the entire codebase.

1. To fix this, it is safer to convert the function to a regular async function.
2. In the corrected code:
3. We define an async function handlingK3 that takes a parameter typeid.
4. We convert the typeid to a number using Number() and store it in a variable game.
5. We then await the funHanding function passing in the game variable.
6. Finally, we await the plusMoney function passing in the game variable.

**listOrderOld**

A function that retrieves a list of old K3 game orders.

*const* listOrderOld = *async* (*req*, *res*) *=>* {

*let* { gameJoin, pageno, pageto } = *req*.body;

*let* auth = *req*.cookies.auth;

*let* checkGame = ['1', '3', '5', '10'].includes(*String*(gameJoin));

    if (!checkGame || pageno < 0 || pageto < 0) {

        return *res*.status(200).json({

            code: 0,

            msg: "Không còn dữ liệu",

            data: {

                gameslist: [],

            },

            status: false

        });

    }

*const* [user] = await connection.query('SELECT `phone`, `code`, `invite`, `level`, `money` FROM users WHERE token = ? AND veri = 1  LIMIT 1 ', [auth]);

*let* game = *Number*(gameJoin);

*const* [k5d] = await connection.query(`SELECT \* FROM k3 WHERE status != 0 AND game = '${game}' ORDER BY id DESC LIMIT ${pageno}, ${pageto} `);

*const* [k5dAll] = await connection.query(`SELECT \* FROM k3 WHERE status != 0 AND game = '${game}' `);

*const* [period] = await connection.query(`SELECT period FROM k3 WHERE status = 0 AND game = '${game}' ORDER BY id DESC LIMIT 1 `);

    if (k5d.length == 0) {

        return *res*.status(200).json({

            code: 0,

            msg: "No more data",

            data: {

                gameslist: [],

            },

            page: 1,

            status: false

        });

    }

    if (!pageno || !pageto || !user[0] || !k5d[0] || !period[0]) {

        return *res*.status(200).json({

            message: 'Error!',

            status: false

        });

    }

*let* page = Math.ceil(k5dAll.length / 10);

    return *res*.status(200).json({

        code: 0,

        msg: "Get success",

        data: {

            gameslist: k5d,

        },

        period: period[0].period,

        page: page,

        status: true

    });

}

This code defines an asynchronous function listOrderOld that takes req and res as parameters.

1. It extracts gameJoin, pageno, and pageto from the request body and auth from cookie.
2. It checks if gameJoin is one of '1', '3', '5', or '10', and if pageno and pageto are greater than 0. If not, it returns a JSON response with specific messages.
3. Then it executes database queries to fetch user information, game details, and periods based on the provided parameters. It handles various edge cases like empty results and missing required data.
4. Finally, the function calculates the total number of pages based on the fetched data and returns a JSON response with the appropriate data, including game details, periods, and status.

**GetMyEmerdList**

A function that retrieves the list of K3 game orders for the logged-in user.

*const* GetMyEmerdList = *async* (*req*, *res*) *=>* {

*let* { gameJoin, pageno, pageto } = *req*.body;

*let* auth = *req*.cookies.auth;

*let* checkGame = ['1', '3', '5', '10'].includes(*String*(gameJoin));

    if (!checkGame || pageno < 0 || pageto < 0) {

        return *res*.status(200).json({

            code: 0,

            msg: "No more data",

            data: {

                gameslist: [],

            },

            status: false

        });

    }

*let* game = *Number*(gameJoin);

*const* [user] = await connection.query('SELECT `phone`, `code`, `invite`, `level`, `money` FROM users WHERE token = ? AND veri = 1 LIMIT 1 ', [auth]);

*const* [result\_5d] = await connection.query(`SELECT \* FROM result\_k3 WHERE phone = ? AND game = '${game}' ORDER BY id DESC LIMIT ${*Number*(pageno) + ',' + *Number*(pageto)}`, [user[0].phone]);

*const* [result\_5dAll] = await connection.query(`SELECT \* FROM result\_k3 WHERE phone = ? AND game = '${game}' ORDER BY id DESC `, [user[0].phone]);

    if (!result\_5d[0]) {

        return *res*.status(200).json({

            code: 0,

            msg: "No more data",

            data: {

                gameslist: [],

            },

            page: 1,

            status: false

        });

    }

    if (!pageno || !pageto || !user[0] || !result\_5d[0]) {

        return *res*.status(200).json({

            message: 'Error!',

            status: true

        });

    }

*let* page = Math.ceil(result\_5dAll.length / 10);

*let* datas = result\_5d.map((*data*) *=>* {

*let* { id, phone, code, invite, level, game, ...others } = *data*;

        return others;

    });

    return *res*.status(200).json({

        code: 0,

        msg: "Get success",

        data: {

            gameslist: datas,

        },

        page: page,

        status: true

    });

}

The provided code is an asynchronous function named GetMyEmerdList that fetches and processes data based on the request body and cookies.

Here is an explanation of the code:

1. It extracts gameJoin, pageno, and pageto from the request body and auth from the cookies.
2. It checks if the gameJoin value is valid and the page numbers are not negative.
3. It performs database queries to fetch user data and game results based on the provided parameters.
4. If certain conditions are not met or data is not found, appropriate error responses are sent.
5. The fetched results are processed to extract necessary data and calculate the total number of pages.
6. Finally, it constructs a JSON response with the processed data.

**distributeCommission**

A function that distributes commissions to users based on their K3 game bets.

*const* distributeCommission = *async* () *=>* {

      try {

*const* { startOfYesterdayTimestamp, endOfYesterdayTimestamp } =

          yesterdayTime();

*const* [levelResult] = await connection.query("SELECT f1 FROM level");

*const* levels = levelResult.map((*row*) *=>* *row*.f1 / 100);

        // const [bets] = await connection.query('SELECT phone, SUM(money + fee) AS total\_money FROM minutes\_1 WHERE time > ? AND time <= ? GROUP BY phone', [startOfDay, endTime]);

*const* [bets] = await connection.query(

          `

          SELECT phone, SUM(total\_money) AS total\_money

          FROM (

            SELECT phone, SUM(money + fee) AS total\_money

            FROM result\_k3

            WHERE time > ? AND time <= ?

            GROUP BY phone

            UNION ALL

            SELECT phone, SUM(money + fee) AS total\_money

            FROM result\_k3

            WHERE time > ? AND time <= ?

            GROUP BY phone

          ) AS combined

          GROUP BY phone

          `,

          [

            startOfYesterdayTimestamp,

            endOfYesterdayTimestamp,

            startOfYesterdayTimestamp,

            endOfYesterdayTimestamp,

          ],

        );

*const* promises = bets.map((*bet*) *=>*

          rosesPlus1(*bet*.phone, *bet*.total\_money, levels, endOfYesterdayTimestamp),

        );

*const* response = await *Promise*.all(promises);

        return {

          success: true,

          message: "Commissions distributed successfully.",

        };

      } catch (error) {

        console.error(error);

        return { success: false, message: error.message };

      }

    };

The provided code defines an asynchronous function distributeCommission that is responsible for distributing commissions. Here's a breakdown of its functionality:

1. It retrieves startOfYesterdayTimestamp and endOfYesterdayTimestamp using a function yesterdayTime.
2. It queries the database for f1 from a table called level and extracts it into levelResult array. Then it maps a new array levels by dividing each f1 value by 100.
3. It constructs a complex SQL query to retrieve total bets for each player for a specific time range and stores the result in bets array.
4. It creates an array of promises by mapping over bets and calling a function rosesPlus1 with relevant parameters.
5. It awaits all the promises to resolve using Promise.all and returns a success message if everything went well; otherwise, it catches any errors and logs them.

**Helper Functions**

## isNumber(params)

This function checks if the given parameter is a number. It uses a regular expression pattern to test if the parameter consists of only digits.

### Parameters:

* params: The parameter to be checked for number validity.

### Return Value:

A boolean value indicating whether the parameter is a number or not.

*const* isNumber = (*params*) *=>* {

*let* pattern = /^[0-9]\*\d$/;

    return pattern.test(*params*);

}

The provided code defines a function isNumber that takes a single parameter params and checks if the input is a number. Here's a breakdown of the code:

1. const isNumber = (params) => { ... }: This line defines an arrow function called isNumber that takes a parameter params.
2. let pattern = /^[0-9]\*\d$/;: This line declares a regular expression pattern that matches a number. The pattern /^[0-9]\*\d$/ checks if the input contains only numeric characters.
3. return pattern.test(params);: This line uses the test method of the regular expression object to test if the params match the number pattern. It returns a boolean value indicating whether the input is a number or not.

## formateT(params)

This function formats a given parameter as a two-digit number. If the parameter is less than 10, a leading zero is added to it.

### Parameters:

* params: The parameter to be formatted.

### Return Value:

A formatted string representing the parameter as a two-digit number.

The provided code defines a function called formateT that takes an integer parameter params and returns a formatted string. Here's the explanation of the code:

1. The initial code checks if the input parameter params is less than 10.
2. If it is less than 10, it concatenates a '0' to the params and stores the result in the variable result.
3. Finally, it returns the result as a string.
4. In the optimized code, I made a few corrections:
5. Added the strconv package for converting integers to strings.
6. Specified the data type for the input parameter params as int.
7. Changed the logic to convert params to a string using strconv.Itoa method.

## timerJoin(params = '', addHours = 0)

This function generates a formatted date and time string. It takes an optional parameter representing a timestamp and an optional parameter representing the number of hours to be added to the current time. If no parameters are provided, the current date and time are used.

### Parameters:

* params (optional): The timestamp to be used for generating the date and time.
* addHours (optional): The number of hours to be added to the date and time.

### Return Value:

A string representing the formatted date and time in the format 'YYYY-MM-DD HH:MM:SS AM/PM'.

*function* timerJoin(*params* = '', *addHours* = 0) {

*let* date = '';

        if (*params*) {

            date = new *Date*(*Number*(*params*));

        } else {

            date = new *Date*();

        }

        date.setHours(date.getHours() + *addHours*);

*let* years = formateT(date.getFullYear());

*let* months = formateT(date.getMonth() + 1);

*let* days = formateT(date.getDate());

*let* hours = date.getHours() % 12;

        hours = hours === 0 ? 12 : hours;

*let* ampm = date.getHours() < 12 ? "AM" : "PM";

*let* minutes = formateT(date.getMinutes());

*let* seconds = formateT(date.getSeconds());

        return years + '-' + months + '-' + days + ' ' + hours + ':' + minutes + ':' + seconds + ' ' + ampm;

    }

The given code defines a function timerJoin that takes two parameters: params (which defaults to an empty string) and addHours (which defaults to 0).

1. It then initializes a variable date to an empty string. If params is provided, it converts params to a number and creates a new Date object using that value. If params is not provided, it creates a new Date object with the current date and time.
2. The function then adds the number of hours specified by addHours to the date. It extracts the year, month, day, hours (in 12-hour format with AM/PM), minutes, and seconds from the date.
3. Finally, it formats these components and returns a string representing the date and time in the format: YYYY-MM-DD HH:MM:SS AM/PM.

## rosesPlus(auth, money)

This function performs a series of database queries and updates to calculate and update the 'roses' value for a user based on their 'money' value and a hierarchical structure determined by user invitations and levels. The function first fetches the 'level' data from the database. Then it fetches the user information based on the provided authentication token. It calculates and updates the 'roses' values for the user and their respective levels of invitations.

### Parameters:

* auth: The authentication token of the user.
* money: The money value associated with the user.

*const* rosesPlus = *async* (*auth*, *money*) *=>* {

*const* [level] = await connection.query('SELECT \* FROM level ');

*let* level0 = level[0];

*const* [user] = await connection.query('SELECT `phone`, `code`, `invite` FROM users WHERE token = ? AND veri = 1  LIMIT 1 ', [*auth*]);

*let* userInfo = user[0];

*const* [f1] = await connection.query('SELECT `phone`, `code`, `invite`, `rank` FROM users WHERE code = ? AND veri = 1  LIMIT 1 ', [userInfo.invite]);

    if (*money* >= 10000) {

        if (f1.length > 0) {

*let* infoF1 = f1[0];

*let* rosesF1 = (*money* / 100) \* level0.f1;

*const* sql6 = `

            INSERT INTO turn\_over (phone, code, invite, daily\_turn\_over, total\_turn\_over)

            VALUES (?, ?, ?, ?, ?)

            ON DUPLICATE KEY UPDATE

            daily\_turn\_over = daily\_turn\_over + VALUES(daily\_turn\_over),

            total\_turn\_over = total\_turn\_over + VALUES(total\_turn\_over)

            `;

            await connection.execute(sql6, [infoF1.phone, infoF1.code, infoF1.invite, *money*, *money*]);

            await connection.query('UPDATE users SET money = money + ?, roses\_f1 = roses\_f1 + ?, roses\_f = roses\_f + ?, roses\_today = roses\_today + ? WHERE phone = ? ', [rosesF1, rosesF1, rosesF1, rosesF1, infoF1.phone]);

*const* [f2] = await connection.query('SELECT `phone`, `code`, `invite`, `rank` FROM users WHERE code = ? AND veri = 1  LIMIT 1 ', [infoF1.invite]);

            if (f2.length > 0) {

*let* infoF2 = f2[0];

*let* rosesF2 = (*money* / 100) \* level0.f2;

                await connection.query('UPDATE users SET money = money + ?, roses\_f = roses\_f + ?, roses\_today = roses\_today + ? WHERE phone = ? ', [rosesF2, rosesF2, rosesF2, infoF2.phone]);

*const* [f3] = await connection.query('SELECT `phone`, `code`, `invite`, `rank` FROM users WHERE code = ? AND veri = 1  LIMIT 1 ', [infoF2.invite]);

                if (f3.length > 0) {

*let* infoF3 = f3[0];

*let* rosesF3 = (*money* / 100) \* level0.f3;

                    await connection.query('UPDATE users SET money = money + ?, roses\_f = roses\_f + ?, roses\_today = roses\_today + ? WHERE phone = ? ', [rosesF3, rosesF3, rosesF3, infoF3.phone]);

*const* [f4] = await connection.query('SELECT `phone`, `code`, `invite`, `rank` FROM users WHERE code = ? AND veri = 1  LIMIT 1 ', [infoF3.invite]);

                    if (f4.length > 0) {

*let* infoF4 = f4[0];

*let* rosesF4 = (*money* / 100) \* level0.f4;

                        await connection.query('UPDATE users SET money = money + ?, roses\_f = roses\_f + ?, roses\_today = roses\_today + ? WHERE phone = ? ', [rosesF4, rosesF4, rosesF4, infoF4.phone]);

                    }

                }

            }

        }

    }

}

### Example:

await rosesPlus('exampleToken', 15000);

The provided code defines an asynchronous function rosesPlus that takes two parameters auth and money. It fetches data from a database table using queries based on certain conditions and updates the data accordingly. Here's a breakdown of the code:

1. It retrieves the level details from the level table and the user information based on the auth token.
2. If the money value is greater than or equal to 10000, it proceeds with further operations.
3. It calculates the amount of roses (rosesF1, rosesF2, rosesF3, rosesF4) based on the money value and specific levels.
4. It inserts or updates the turn\_over table based on certain conditions and increments the turn-over values.
5. It updates the users table with the calculated rose amounts and adjusts the money and rose values for various users (infoF1, infoF2, infoF3, infoF4) hierarchically based on the invitation structure.
6. This code essentially performs database operations to handle transactions and update user data based on certain business logic.

## validateBet(join, list\_join, x, money, game)

This function validates a user's bet based on several conditions. It checks if the 'join' parameter is one of the allowed values ('a', 'b', 'c', 'd', 'e', 'total'). It also checks the length of the 'list\_join' parameter, ensuring it does not exceed 10 characters. Additionally, it verifies if the 'x' parameter and the 'money' parameter are numbers, and if the 'game' parameter is one of the allowed values ('1', '3', '5', '10'). If any of these conditions fail, the function returns false. Otherwise, it checks the content of the 'list\_join' parameter, validating it based on whether it contains only numbers or specific characters ('c', 'l', 'b', 's').

### Parameters:

* join: The type of bet the user wants to place.
* list\_join: The specific bets the user wants to place.
* x: The value associated with the bet.
* money: The amount of money the user wants to bet.
* game: The specific game the user wants to participate in.

### Return Value:

A boolean value indicating whether the bet is valid or not.

*const* validateBet = *async* (*join*, *list\_join*, *x*, *money*, *game*) *=>* {

*let* checkJoin = isNumber(*list\_join*);

*let* checkX = isNumber(*x*);

*const* checks = ['a', 'b', 'c', 'd', 'e', 'total'].includes(*join*);

*const* checkGame = ['1', '3', '5', '10'].includes(*String*(*game*));

*const* checkMoney = ['1000', '10000', '100000', '1000000'].includes(*money*);

    if (!checks || *list\_join*.length > 10 || !checkX || !checkMoney || !checkGame) {

        return false;

    }

    if (checkJoin) {

*let* arr = *list\_join*.split('');

*let* length = arr.length;

        for (*let* i = 0; i < length; i++) {

*const* joinNum = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'].includes(arr[i]);

            if (!joinNum) {

                return false;

            }

        }

    } else {

*let* arr = *list\_join*.split('');

*let* length = arr.length;

        for (*let* i = 0; i < length; i++) {

*const* joinStr = ["c", "l", "b", "s"].includes(arr[i]);

            if (!joinStr) {

                return false;

            }

        }

    }

    return true;

}

In the provided code snippet, there is a JavaScript function validateBet defined using async arrow function syntax.

1. This function takes five parameters: join, list\_join, x, money, and game.
2. The function first checks if the parameters meet certain conditions to determine the validity of a bet. It uses helper functions like isNumber (not defined in the snippet) to check if list\_join and x are numbers, then performs various checks based on the values of join, game, and money.
3. If any of the conditions are not met, the function returns false. Otherwise, it splits the list\_join string into an array and validates each element based on whether it is a number or belongs to a predefined set of characters ('c', 'l', 'b', 's'). If all validations pass, the function returns true.

### Example:

const result = await validateBet('a', '12345', 2, '10000', '1'); // true

## makeid(length)

This function generates a random alphanumeric ID of the specified length.

### Parameters:

* length (number): The length of the ID to be generated.

### Returns:

A string representing the randomly generated ID.

*function* makeid(*length*) {

*var* result = '';

*var* characters = '123456';

*var* charactersLength = characters.length;

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

        result += characters.charAt(Math.floor(Math.random() \* charactersLength));

    }

    return result;

}

In this code snippet, a function named makeid is defined that takes a parameter length indicating the desired length of the generated ID.

1. The function generates a random ID using characters 123456.
2. The code uses the math/rand package to generate random numbers. It initializes the random number generator with a seed based on the current time to ensure different results on each run.
3. Inside the loop, the function randomly selects a character from the characters string and appends it to the result string. Finally, the generated random ID is returned.

## funHanding(game)

This asynchronous function handles the game logic for the specified game. It retrieves the latest game data from the database, updates the game result, and performs various calculations based on the game rules.

### Parameters:

* game (number): The identifier of the game.

### Returns:

This function does not have a return value.

*async* *function* funHanding(*game*) {

*const* [k5d] = await connection.query(`SELECT \* FROM k3 WHERE status != 0 AND game = ${*game*} ORDER BY id DESC LIMIT 1 `);

*let* k5dInfo = k5d[0];

    // update ket qua

    await connection.execute(`UPDATE result\_k3 SET result = ? WHERE status = 0 AND game = ${*game*}`, [k5dInfo.result]);

*let* result = *String*(k5dInfo.result).split('');

*let* total = 0;

    for (*let* i = 0; i < result.length; i++) {

        total += *Number*(result[i]);

    }

    // xử lý game Tổng số

*const* [totalNumber] = await connection.execute(`SELECT id, bet FROM result\_k3 WHERE status = 0 AND game = ${*game*} AND typeGame = 'total' `);

*let* totalN = totalNumber.length;

    for (*let* i = 0; i < totalN; i++) {

*let* sult = totalNumber[i].bet.split(',');

        // let result = sult.includes(String(total));

        // if (!result) {

        //     await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [totalNumber[i].id]);

        // }

*let* lengWin = sult.filter(*function*(*age*) {

            return *age* == total;

        });

*let* check1 = false;

*let* check2 = false;

*let* check3 = false;

*let* check4 = false;

        if (total >= 3 && total <= 10 && sult.includes('s')) {

            check1 = true;

        } else {

            check1 = false;

        }

        if (total >= 11 && total <= 18 && sult.includes('b')) {

            check2 = true;

        } else {

            check2 = false;

        }

        if (total % 2 == 0 && sult.includes('c')) {

            check3 = true;

        } else {

            check3 = false;

        }

        if (total % 2 != 0 && sult.includes('l')) {

            check4 = true;

        } else {

            check4 = false;

        }

        if (!check1 && !check2 && !check3 && !check4) {

            await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [totalNumber[i].id]);

        }

        if (lengWin.length >= 1) {

            await connection.execute(`UPDATE result\_k3 SET status = 0 WHERE id = ? `, [totalNumber[i].id]);

        }

    }

    // xử lý game 2 số trùng nhau

*const* [totaltwoSame] = await connection.execute(`SELECT id, bet FROM result\_k3 WHERE status = 0 AND game = ${*game*} AND typeGame = 'two-same' `);

*let* totalTwoSame = totaltwoSame.length;

    for (*let* i = 0; i < totalTwoSame; i++) {

*let* sult = totaltwoSame[i].bet.split('@');

*let* id = totaltwoSame[i].id;

        if (sult[0].length > 0 && sult[1].length > 0) {

*let* array = sult[0].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* check1 = false;

*let* check2 = false;

*let* array2 = sult[1].split('&');

            for (*let* i = 0; i < array2.length; i++) {

*let* newArr = array2[i].split('|')[0];

*let* newArr2 = array2[i].split('|')[1].split(',');

*let* resultA1 = newArr.includes(*String*(kq1));

*let* resultA2 = newArr.includes(*String*(kq2));

                if (!resultA1 && !resultA2) {

                    check1 = true;

                } else if (resultA1 && !resultA2) {

*let* resultA4 = newArr2.includes(*String*(kq[2]));

                    if (!resultA4) {

                        check1 = true;

                    }

                } else if (!resultA1 && resultA2) {

*let* resultA3 = newArr2.includes(*String*(kq[0]));

                    if (!resultA3) {

                        check1 = true;

                    }

                }

            }

*let* result1 = array.includes(*String*(kq1));

*let* result2 = array.includes(*String*(kq2));

            if (!result1 && !result2) {

                check2 = true;

            }

            if (check1 && check2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length > 0 && sult[1].length <= 0) {

*let* array = sult[0].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* result1 = array.includes(*String*(kq1));

*let* result2 = array.includes(*String*(kq2));

            if (!result1 && !result2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length <= 0 && sult[1].length > 0) {

*let* kq = *String*(k5dInfo.result).split('');

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* kq3 = kq[0];

*let* kq4 = kq[2];

*let* check = false;

*let* array2 = sult[1].split('&');

            for (*let* i = 0; i < array2.length; i++) {

*let* newArr = array2[i].split('|')[0];

*let* newArr2 = array2[i].split('|')[1].split(',');

*let* resultA1 = newArr.includes(*String*(kq1));

*let* resultA2 = newArr.includes(*String*(kq2));

*let* resultA3 = newArr2.includes(*String*(kq3));

*let* resultA4 = newArr2.includes(*String*(kq4));

                if (!resultA1 && !resultA2) {

                    if (!resultA3 && !resultA4) {

                        await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

                    }

                } else if (resultA1 && !resultA2) {

*let* resultA4 = newArr2.includes(*String*(kq[2]));

                    if (resultA4) {

                        check = true;

                    }

                } else if (!resultA1 && resultA2) {

*let* resultA3 = newArr2.includes(*String*(kq[0]));

                    if (resultA3) {

                        check = true;

                    }

                }

            }

            if (!check) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        }

    }

    // xử lý game 3 số trùng nhau

*const* [ThreeSame] = await connection.execute(`SELECT id, bet FROM result\_k3 WHERE status = 0 AND game = ${*game*} AND typeGame = 'three-same' `);

*let* ThreeSameL = ThreeSame.length;

    for (*let* i = 0; i < ThreeSameL; i++) {

*let* sult = ThreeSame[i].bet.split('@');

*let* id = ThreeSame[i].id;

        if (sult[0].length > 0 && sult[1].length > 0) {

*let* array = sult[0].split(',');

*let* kq = *String*(k5dInfo.result);

*let* check1 = false;

*let* check2 = false;

*let* resultA1 = array.includes(*String*(kq));

            if (!resultA1) {

                check1 = true;

            }

*let* result = ['111', '222', '333', '444', '555', '666'].includes(*String*(kq));

            if (!result) {

                check2 = true;

            }

            if (check1 && check2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length > 0 && sult[1].length <= 0) {

*let* array = sult[0].split(',');

*let* kq = *String*(k5dInfo.result);

*let* result = array.includes(*String*(kq));

            if (!result) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length <= 0 && sult[1].length > 0) {

*let* kq = *String*(k5dInfo.result);

*let* result = ['111', '222', '333', '444', '555', '666'].includes(*String*(kq));

            if (!result) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        }

    }

    // xử lý game 3 số khác nhau

*const* [Unlike] = await connection.execute(`SELECT id, bet FROM result\_k3 WHERE status = 0 AND game = ${*game*} AND typeGame = 'unlike' `);

*let* Unlikes = Unlike.length;

    for (*let* i = 0; i < Unlikes; i++) {

*let* sult = Unlike[i].bet.split('@');

*let* id = Unlike[i].id;

        if (sult[0].length > 1 && sult[1] == 'u' && sult[2].length > 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

*let* check2 = false;

*let* check3 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* kq3 = kq[0] + kq[2];

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq1));

*let* resultA2 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq2));

*let* resultA3 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq3));

                if (resultA1 || resultA2 || resultA3) {

                    check3 = true;

                }

            }

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array2.includes(*String*(kq[i]));

                if (resultA1) {

                    check2 = true;

                }

            }

            if (check1 && check2 && check3) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length > 1 && sult[1] == 'y' && sult[2].length > 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

*let* check2 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array2.includes(*String*(kq[i]));

                if (resultA1) {

                    check2 = true;

                }

            }

            if (check1 && check2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length > 1 && sult[1] == 'u' && sult[2].length <= 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

*let* check2 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* kq3 = kq[0] + kq[2];

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq1));

*let* resultA2 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq2));

*let* resultA3 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq3));

                if (resultA1 || resultA2 || resultA3) {

                    check2 = true;

                }

            }

            if (check1 && check2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length <= 1 && sult[1] == 'u' && sult[2].length > 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

*let* check2 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array2.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* kq3 = kq[0] + kq[2];

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq1));

*let* resultA2 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq2));

*let* resultA3 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq3));

                if (resultA1 || resultA2 || resultA3) {

                    check2 = true;

                }

            }

            if (check1 && check2) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length > 1 && sult[1] == 'y' && sult[2].length <= 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

            if (check1) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length <= 1 && sult[1] == 'u' && sult[2].length <= 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* kq3 = kq[0] + kq[2];

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq1));

*let* resultA2 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq2));

*let* resultA3 = ['11', '22', '33', '44', '55', '66'].includes(*String*(kq3));

                if (resultA1 || resultA2 || resultA3) {

                    check1 = true;

                }

            }

            if (check1) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        } else if (sult[0].length <= 1 && sult[1] == 'y' && sult[2].length > 1) {

*let* array = sult[0].split(',');

*let* array2 = sult[2].split(',');

*let* kq = *String*(k5dInfo.result).split('');

*let* check1 = false;

            for (*let* i = 0; i < kq.length; i++) {

*let* resultA1 = array2.includes(*String*(kq[i]));

                if (resultA1) {

                    check1 = true;

                }

            }

            if (check1) {

                await connection.execute(`UPDATE result\_k3 SET status = 2 WHERE id = ? `, [id]);

            }

        }

    }

}

The code defines an asynchronous function that processes game results from a database. Here's a simple breakdown:

1. It retrieves the latest game result from the database.
2. It updates the result in another table based on the game's status.
3. It calculates the total of the digits in the result.
4. It checks different game types (like total numbers, pairs, and triples) to see if bets match the results:
   * For each type, it checks specific conditions based on the bets made.
   * It updates the status of bets in the database depending on whether they win or lose.
5. The function handles various game scenarios like two same numbers, three same numbers, and different numbers, updating the status accordingly.
6. Overall, it processes bets and results, updating the database based on the game's logic.

## plusMoney(game)

This asynchronous function calculates and adds the prize money to the user's account for the specified game. It retrieves the game data from the database and performs calculations based on the game type and user's bets.

### Parameters:

* game (number): The identifier of the game.

### Returns:

This function does not have a return value.

*async* *function* plusMoney(*game*) {

*const* [order] = await connection.execute(`SELECT id, phone, bet, price, money, fee, amount, result, typeGame FROM result\_k3 WHERE status = 0 AND game = ${*game*} `);

    for (*let* i = 0; i < order.length; i++) {

*let* orders = order[i];

*let* phone = orders.phone;

*let* id = orders.id;

*let* nhan\_duoc = 0;

*let* result = orders.result;

        if (orders.typeGame == "total") {

*let* arr = orders.bet.split(',');

*let* totalResult = orders.result.split('');

*let* totalResult2 = 0;

            for (*let* i = 0; i < 3; i++) {

                totalResult2 += *Number*(totalResult[i]);

            }

*let* total = (orders.money / arr.length);

*let* fee = total \* 0.02;

*let* price = total - fee;

*let* lengWin = arr.filter(*function*(*age*) {

                return *age* == totalResult2;

            });

*let* lengWin2 = arr.filter(*function*(*age*) {

                return !isNumber(*age*);

            });

            if (totalResult2 % 2 == 0 && lengWin2.includes('c')) {

                nhan\_duoc += price \* 1.92;

            }

            if (totalResult2 % 2 != 0 && lengWin2.includes('l')) {

                nhan\_duoc += price \* 1.92;

            }

            if (totalResult2 >= 11 && totalResult2 <= 18 && lengWin2.includes('b')) {

                nhan\_duoc += price \* 1.92;

            }

            if (totalResult2 >= 3 && totalResult2 <= 11 && lengWin2.includes('s')) {

                nhan\_duoc += price \* 1.92;

            }

*let* get = 0;

            switch (lengWin[0]) {

                case '3':

                    get = 207.36;

                    break;

                case '3':

                    get = 69.12;

                    break;

                case '5':

                    get = 34.56;

                    break;

                case '6':

                    get = 20.74;

                    break;

                case '7':

                    get = 13.83;

                    break;

                case '8':

                    get = 9.88;

                    break;

                case '9':

                    get = 8.3;

                    break;

                case '10':

                    get = 7.68;

                    break;

                case '11':

                    get = 7.68;

                    break;

                case '12':

                    get = 8.3;

                    break;

                case '13':

                    get = 9.88;

                    break;

                case '14':

                    get = 13.83;

                    break;

                case '15':

                    get = 20.74;

                    break;

                case '16':

                    get = 34.56;

                    break;

                case '17':

                    get = 69.12;

                    break;

                case '18':

                    get = 207.36;

                    break;

            }

            nhan\_duoc += price \* get;

            await connection.execute('UPDATE `result\_k3` SET `get` = ?, `status` = 1 WHERE `id` = ? ', [nhan\_duoc, id]);

*const* sql = 'UPDATE `users` SET `money` = `money` + ? WHERE `phone` = ? ';

            await connection.execute(sql, [nhan\_duoc, phone]);

        }

        nhan\_duoc = 0;

        if (orders.typeGame == "two-same") {

*let* kq = result.split('');

*let* kq1 = kq[0] + kq[1];

*let* kq2 = kq[1] + kq[2];

*let* array = orders.bet.split('@');

*let* arr1 = array[0].split(',');

*let* arr2 = array[1];

*let* arr3 = array[1].split('&');

            for (*let* i = 0; i < arr1.length; i++) {

                if(arr1[i] != "") {

*let* check1 = arr1[i].includes(kq1);

*let* check2 = arr1[i].includes(kq2);

                    if (check1 || check2) {

*let* lengthArr = 0;

*let* count = 0;

                        if (arr2.length > 0) {

*let* arr = arr2.split('&');

                            for (*let* i = 0; i < arr.length; i++) {

                                arr2 = arr[i].split('|');

                                count = arr2[1].split(',').length;

                            }

                            lengthArr = arr.length;

                            count = count;

                        }

*let* total = orders.money / orders.amount / (lengthArr + arr1.length);

                        nhan\_duoc += total \* 12.83;

                    }

                }

            }

            arr2 = array[1];

            for (*let* i = 0; i < arr3.length; i++) {

                if(arr3[i] != "") {

*let* files = arr3[i].split('|');

*let* check1 = files[0].includes(kq1);

*let* check2 = files[0].includes(kq2);

                    if (check1 || check2) {

*let* lengthArr = 0;

*let* count = 0;

                        if (arr2.length > 0) {

*let* arr = arr2.split('&');

                            for (*let* i = 0; i < arr.length; i++) {

                                arr2 = arr[i].split('|');

                                count = arr2[1].split(',').length;

                            }

                            lengthArr = arr.length;

                            count = count;

                        }

*let* bale = 0;

                        for (*let* i = 0; i < arr1.length; i++) {

                            if (arr1[i] != "") {

                                bale = arr1.length;

                            }

                        }

*let* total = orders.money / orders.amount / (lengthArr + bale);

                        nhan\_duoc += total \* 69.12;

                    }

                }

            }

            nhan\_duoc -= orders.fee;

            await connection.execute('UPDATE `result\_k3` SET `get` = ?, `status` = 1 WHERE `id` = ? ', [nhan\_duoc, id]);

*const* sql = 'UPDATE `users` SET `money` = `money` + ? WHERE `phone` = ? ';

            await connection.execute(sql, [nhan\_duoc, phone]);

        }

        nhan\_duoc = 0;

        if (orders.typeGame == "three-same") {

*let* kq = result;

*let* array = orders.bet.split('@');

*let* arr1 = array[0].split(',');

*let* arr2 = array[1];

            for (*let* i = 0; i < arr1.length; i++) {

                if (arr1[i] != "") {

*let* check1 = arr1[i].includes(kq);

*let* bala = 0;

                    if (arr2 != "") {

                        bala = 1;

                    }

                    if (check1) {

*let* total = (orders.money / (arr1.length + bala) / orders.amount);

                        nhan\_duoc += total \* 207.36 - orders.fee;

                    }

                }

            }

            if (arr2 != "") {

*let* bala = 0;

                for (*let* i = 0; i < arr1.length; i++) {

                    if (arr1[i] != "") {

                        bala = arr1.length;

                    }

                }

*let* total = (orders.money / (1 + bala) / orders.amount);

                nhan\_duoc += total \* 34.56 - orders.fee;

            }

            await connection.execute('UPDATE `result\_k3` SET `get` = ?, `status` = 1 WHERE `id` = ? ', [nhan\_duoc, id]);

*const* sql = 'UPDATE `users` SET `money` = `money` + ? WHERE `phone` = ? ';

            await connection.execute(sql, [nhan\_duoc, phone]);

        }

        nhan\_duoc = 0;

        if (orders.typeGame == "unlike") {

*let* kq = result.split('');

*let* array = orders.bet.split('@');

*let* arr1 = array[0].split(',');

*let* arr2 = array[1];

*let* arr3 = array[2].split(',');

            for (*let* i = 0; i < arr1.length; i++) {

                if (arr1[i] != "") {

*let* check1 = kq.includes(arr1[i]);

*let* bala = 0;

*let* bala2 = 0;

                    for (*let* i = 0; i < arr3.length; i++) {

                        if (arr3[i].length != "") {

                            bala = arr3.length;

                        }

                    }

                    if (arr2 == "u") {

                        bala2 = 1;

                    }

                    if (!check1) {

*let* total = (orders.money / (arr1.length + bala + bala2) / orders.amount);

                        nhan\_duoc += total \* 34.56 - orders.fee;

                        if (arr2 == "u") {

*let* total = (orders.money / (1 + bala + bala2) / orders.amount);

                            nhan\_duoc += (total - orders.fee) \* 8.64;

                        }

                    }

                }

            }

            if (arr2 == "u") {

*let* bala = 0;

*let* bala2 = 0;

                for (*let* i = 0; i < arr1.length; i++) {

                    if (arr1[i] != "") {

                        bala = arr1.length;

                    }

                }

                for (*let* i = 0; i < arr3.length; i++) {

                    if (arr3[i].length != "") {

                        bala2 = arr3.length;

                    }

                }

*let* total = (orders.money / (1 + bala + bala2) / orders.amount);

                nhan\_duoc += (total - orders.fee) \* 8.64;

            }

            for (*let* i = 0; i < arr3.length; i++) {

                if (arr1[i] != "") {

*let* check1 = kq.includes(arr3[i]);

*let* bala = 0;

                    for (*let* i = 0; i < arr1.length; i++) {

                        if (arr1[i].length != "") {

                            bala = arr1.length;

                        }

                    }

                    if (!check1) {

*let* total = (orders.money / (arr3.length + bala) / orders.amount);

                        nhan\_duoc += total \* 6.91 - orders.fee;

                    }

                }

            }

            await connection.execute('UPDATE `result\_k3` SET `get` = ?, `status` = 1 WHERE `id` = ? ', [nhan\_duoc, id]);

*const* sql = 'UPDATE `users` SET `money` = `money` + ? WHERE `phone` = ? ';

            await connection.execute(sql, [nhan\_duoc, phone]);

        }

    }

}

The provided code defines an asynchronous function named plusMoney which takes a parameter game.

1. It retrieves data from a database table named result\_k3 with specific conditions and processes each row of the retrieved data based on the typeGame field.
2. Inside the function, there are conditional blocks for different types of games like 'total', 'two-same', 'three-same', and 'unlike'. Each block contains specific calculations and conditions to update the nhan\_duoc variable.
3. The code also includes multiple loops and conditions to handle different scenarios of the game types and calculate the winnings accordingly. It updates the database tables result\_k3 and users with the calculated winnings for each order.
4. There are some issues in the code such as undefined isNumber function, and repeated variable names within nested loops that need to be addressed.

## rosesPlus1(phone, money, levels = [], timeNow = "")

This asynchronous function calculates and adds bonuses to the user's account based on a referral system. It calculates commissions for the specified levels and updates the users' balances accordingly.

### Parameters:

* phone (string): The phone number of the user.
* money (number): The amount of money to be used for commission calculations.
* levels (array): Optional. An array of commission levels.
* timeNow (string): Optional. The current time in a specific format.

### Returns:

An object with properties success (boolean) and message (string) indicating the success status and any error message.

*const* rosesPlus1 = *async* (*phone*, *money*, *levels* = [], *timeNow* = "") *=>* {

    try {

*const* [userResult] = await connection.query(

        "SELECT `phone`, `code`, `invite`, `money` FROM users WHERE phone = ? AND veri = 1 LIMIT 1",

        [*phone*],

      );

*const* userInfo = userResult[0];

      if (!userInfo) {

        return;

      }

*let* userReferrer = userInfo.invite;

*let* commissionsToInsert = [];

*let* usersToUpdate = [];

*let* timeNow1 = *Date*.now();

      for (*let* i = 0; i < *levels*.length; i++) {

*const* levelCommission = *levels*[i] \* *money*;

*const* [referrerRows] = await connection.query(

          "SELECT phone, money, code, invite FROM users WHERE code = ?",

          [userReferrer],

        );

*const* referrerInfo = referrerRows[0];

        if (referrerInfo) {

*const* commissionId = generateCommissionId();

          commissionsToInsert.push([

            commissionId,

            referrerInfo.phone,

            userInfo.phone,

            levelCommission,

            i + 1,

            timeNow1,

          ]);

          usersToUpdate.push([levelCommission, referrerInfo.phone]);

          userReferrer = referrerInfo.invite;

        } else {

          console.log(`Level ${i + 1} referrer not found.`);

          break;

        }

      }

      if (commissionsToInsert.length > 0) {

        await connection.query(

          "INSERT INTO commissions (commission\_id, phone, from\_user\_phone, money, level, time) VALUES ?",

          [commissionsToInsert],

        );

      }

      if (usersToUpdate.length > 0) {

*const* updatePromises = usersToUpdate.map(([*money*, *phone*]) *=>*

          connection.query("UPDATE users SET money = money + ? WHERE phone = ?", [

*money*,

*phone*,

          ]),

        );

        await *Promise*.all(updatePromises);

      }

      return {

        success: true,

        message: "Commissions calculated and inserted successfully.",

      };

    } catch (error) {

      console.error(error);

      return { success: false, message: error.message };

    }

  };

This code defines an asynchronous function rosesPlus1 that calculates commissions and updates user balances based on a given set of levels and monetary values.

Here is a breakdown of the code:

1. It receives parameters phone, money, levels, and timeNow with default values.
2. It queries the database to retrieve user information based on the provided phone.
3. For each level specified in the levels array, it calculates a commission and updates the referrer's balance accordingly.
4. If a referrer is not found for a specific level, it logs a message and stops further processing.
5. It inserts the calculated commissions into the commissions table and updates the users' balances in the users table.
6. If any errors occur during the process, it catches the error, logs it, and returns a failure message.
7. Finally, it returns a success message if the commissions are calculated and inserted successfully.

## Variables

## priceGet

An object that stores the prize money for different game types and options. It provides easy access to the prize money based on specific keys.

*const* priceGet = {

    total: {

        't3': 207.36,

        't4': 69.12,

        't5': 34.56,

        't6': 20.74,

        't7': 13.83,

        't8': 9.88,

        't9': 8.3,

        't10': 7.68,

        't11': 7.68,

        't12': 8.3,

        't13': 9.88,

        't14': 13.83,

        't15': 20.74,

        't16': 34.56,

        't17': 69.12,

        't18': 207.36,

        'b': 1.92,

        's': 1.92,

        'l': 1.92,

        'c': 1.92,

    },

    two: {

        twoSame: 13.83,

        twoD: 69.12

    },

    three: {

        threeD: 207.36,

        threeSame: 34.56

    },

    unlike: {

        unlikeThree: 34.56,

        threeL: 8.64,

        unlikeTwo: 6.91

    }

}

Note: The above code is provided as a sample and may not reflect the actual functionality of the code in the project. Modify the documentation accordingly to match the actual code implementation.