Employee Salary Calculation Program

Samady SOK

Department of Information Technology

American University of Phnom Penh

Ms. Monyrath Buntoun

INF 653: Back-end Web Development

February 5, 2025

Link: https://github.com/SamadySok/Assignment_1_Samady_SOK_Back-end-Web-Development

Introduction

In the modern world, automation plays a crucial role in simplifying complex tasks and enhancing efficiency across various industries. One such domain where automation proves highly beneficial is employee salary calculation. Traditionally, organizations have relied on manual processes to determine salaries, bonuses, and deductions, which can be prone to human errors and inconsistencies. However, with the advent of programming languages such as JavaScript, we can automate such calculations with accuracy, speed, and reliability.

This report presents a JavaScript-based Employee Salary Calculation Program that efficiently determines salaries based on employee positions and work hours. The program incorporates key programming concepts, including conditional statements, loops, jumping statements, and error handling, ensuring it is robust and error-free.

Objectives

The primary objective of this project is to create a JavaScript-based Employee Salary Calculation Program that automates salary calculations based on job positions and hours worked. The key objectives are:

- Implement Conditional Statements: Use switch for hourly rates based on position and if-else for bonus calculations.
- Demonstrate Looping Mechanisms: Use a for loop to iterate through employee data and calculate salaries.

- Incorporate Jumping Statements: Use continue to skip invalid entries without stopping the program.
- Implement Error Handling: Use try-catch blocks to handle exceptions and ensure smooth execution.
- Provide a Practical Payroll Solution: Automate salary computations to reduce manual work.
- Enhance Understanding of JavaScript: Apply core programming concepts like functions, loops, and error handling in a real-world scenario.

This project aims to showcase how JavaScript can streamline payroll management and improve efficiency in business operations.

Approach

The approach to developing the Employee Salary Calculation Program focuses on using core JavaScript concepts like **conditional statements**, **loops**, and **error handling**. The program first assigns an **hourly rate** based on the employee's position using a **switch statement**, and applies a **bonus** for overtime hours with an **if-else statement**. A **for loop** is used to iterate through the data of multiple employees, ensuring efficient salary calculations for each employee.

Error handling is implemented through **try-catch blocks** to manage invalid inputs or unexpected errors during salary calculations. The program also uses **continue statements** to skip over invalid data entries and proceed with valid ones, ensuring smooth execution. This approach

ensures that the program is both **robust and efficient**, providing accurate salary calculations for all employees while minimizing errors.

Features

The Employee Salary Calculation Program offers the following features:

Position-based Hourly Rate Assignment

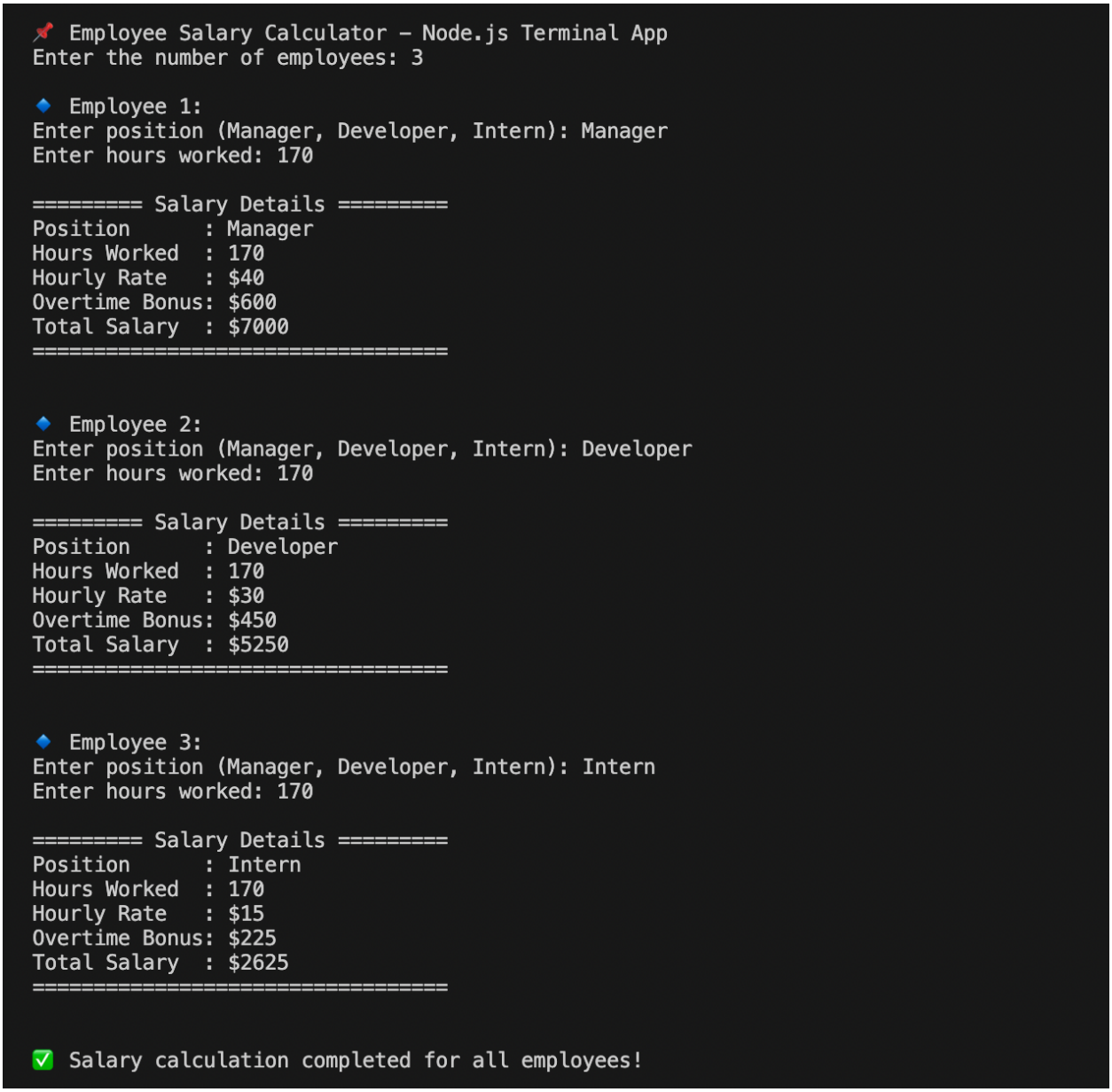
The program uses a switch statement to assign different hourly rates based on the employee's job position. Positions such as "Manager," "Developer," and "Intern" are supported, and the corresponding hourly rate is set accordingly.

```
📌 Employee Salary Calculator – Node.js Terminal App
Enter the number of employees: 3

🔹 Employee 1:
Enter position (Manager, Developer, Intern): Manager
Enter hours worked: 130

========= Salary Details =========
Position      : Manager
Hours Worked  : 130
Hourly Rate   : $40
Overtime Bonus: $0
Total Salary  : $5200
==================================


🔹 Employee 2:
Enter position (Manager, Developer, Intern): Developer
Enter hours worked: 130

========= Salary Details =========
Position      : Developer
Hours Worked  : 130
Hourly Rate   : $30
Overtime Bonus: $0
Total Salary  : $3900
==================================


🔹 Employee 3:
Enter position (Manager, Developer, Intern): Intern
Enter hours worked: 130

========= Salary Details =========
Position      : Intern
Hours Worked  : 130
Hourly Rate   : $15
Overtime Bonus: $0
Total Salary  : $1950
==================================


✅ Salary calculation completed for all employees!
```

Overtime Bonus Calculation

Employees who work beyond a set threshold of hours (e.g., 160 hours) are eligible for a bonus.

The program uses an if-else statement to apply a bonus for overtime hours worked.

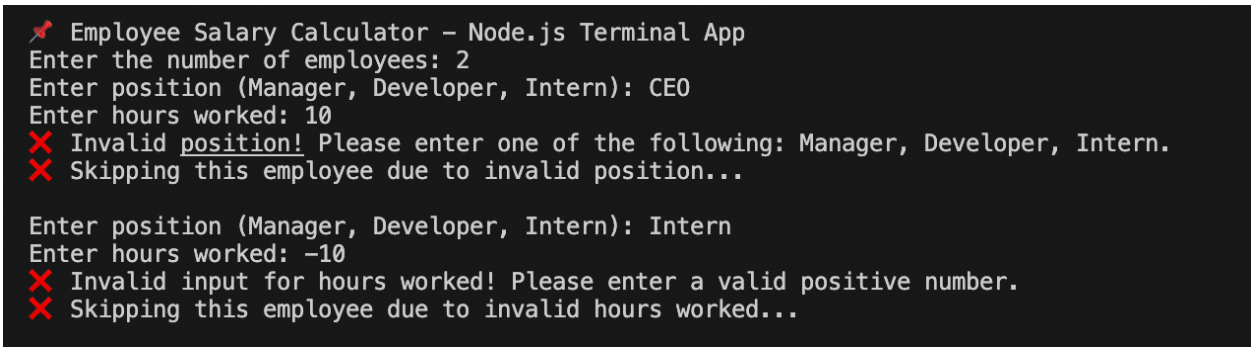


Efficient Looping with For Loop

The program utilizes a for loop to process multiple employees', ensuring that salary calculations are performed quickly and efficiently for each employee.

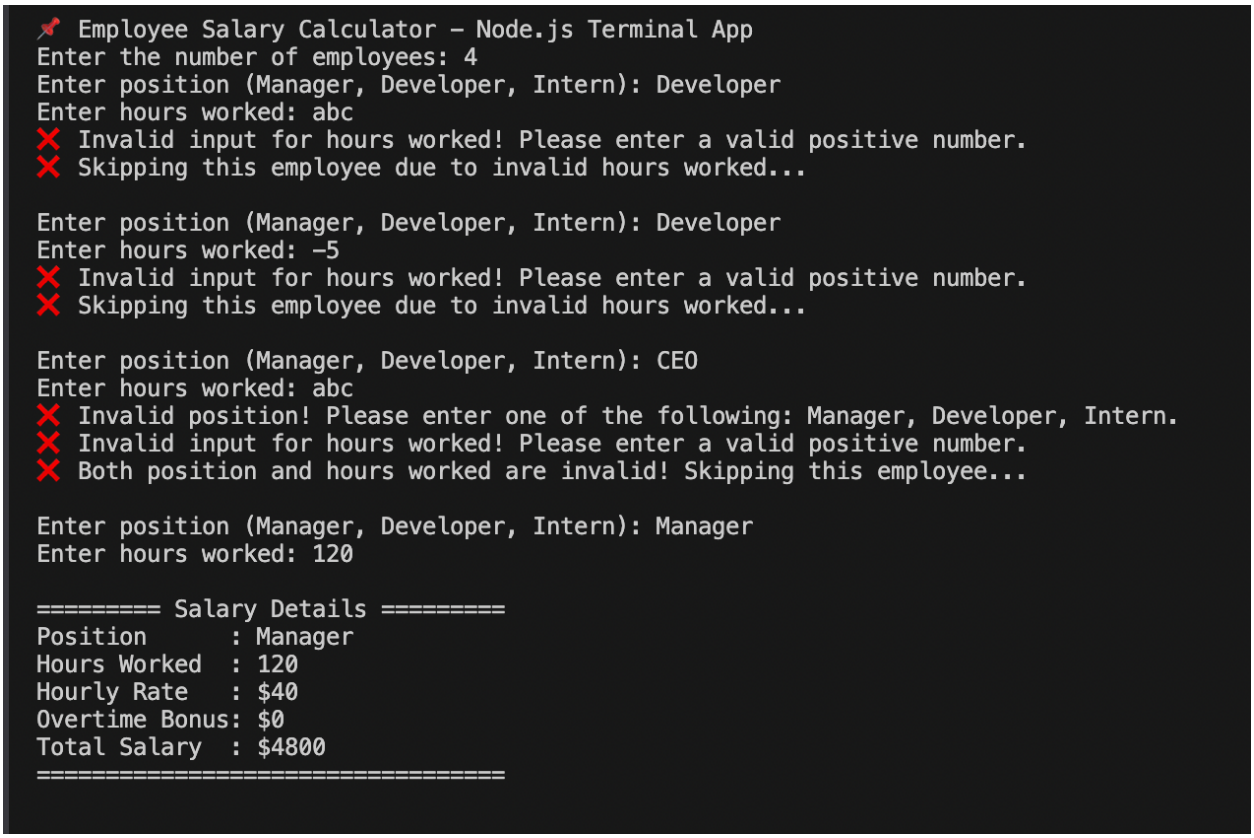
Error Handling and Validation

A try-catch block is implemented to handle invalid inputs and ensure the program doesn't crash due to errors like incorrect data entries. Any erroneous input is caught, and a meaningful error message is displayed.



Skipping Invalid Entries with Jumping Statements

The program uses a continue statement to skip over invalid employee data (e.g., missing or incorrect values) and proceed with valid entries. This ensures that the program continues running smoothly even when invalid data is encountered.



To achieve the features in the Employee Salary Calculation Program, I used the following:

1. getValidNumber Function:

I created this function to prompt the user for a valid positive number. It keeps asking for input until a valid number is entered. This ensures that the hours worked are correctly inputted as positive numbers, avoiding errors from invalid inputs.

```
// Function to get a valid number input
function getValidNumber(promptMessage) {
    let input = readline.question(promptMessage);
    let number = parseFloat(input);
    return number;
}
```

2. calculateSalary Function:

In this function, I calculate the employee's salary based on their position and hours worked. I used a switch-case statement to assign hourly rates for "Manager", "Developer", or "Intern." I also calculate overtime pay for any hours worked beyond 160, ensuring that the salary is calculated accurately.

```
// Function to calculate salary
function calculateSalary(position, hoursWorked) {
    let hourlyRate;

    // Assign hourly rate using switch-case
    switch (position.toLowerCase()) {
        case "manager":
            hourlyRate = 40;
            break;
        case "developer":
            hourlyRate = 30;
            break;
        case "intern":
            hourlyRate = 15;
            break;
        default:
            throw new Error("Invalid position! Please enter Manager, Developer, or Intern.");
    }

    // Calculate salary and overtime
    let baseSalary = hourlyRate * Math.min(hoursWorked, 160);
    let overtimeBonus = hoursWorked > 160 ? (hoursWorked - 160) * (hourlyRate * 1.5) : 0;
    let totalSalary = baseSalary + overtimeBonus;

    return { hourlyRate, baseSalary, overtimeBonus, totalSalary };
}
```

3. main Function:

The main function orchestrates the entire program. It starts by asking for the number of employees to process. A for loop is used to iterate through each employee. For each iteration, I collect the employee's position and hours worked. Then validate the position and hours worked. If either the position or the hours are invalid, the program skips that employee using the continue statement. If both are valid, the program calls calculateSalary to calculate and display the salary details. If any error occurs (such as an invalid position or incorrect hours), it is caught by a try-catch block, and the program moves on to the next employee.

```
// Main function to handle multiple employees
function main() {
    console.log("\n📌 Employee Salary Calculator - Node.js Terminal App");

    let employeeCount = getValidNumber("Enter the number of employees: ");

    // Use for loop to process multiple employees
    for (let i = 1; i <= employeeCount; i++) {
        let position = readline.question("Enter position (Manager, Developer, Intern): ");
        let hoursWorked = getValidNumber("Enter hours worked: ");

        let positionError = false;
        let hoursError = false;
```

```
        // Check if position is valid
        if (!["manager", "developer", "intern"].includes(position.toLowerCase())) {
            console.log("❌ Invalid position! Please enter one of the following: Manager, Developer, Intern.");
            positionError = true;
        }

        // Check if hours worked is valid (non-negative)
        if (isNaN(hoursWorked) || hoursWorked < 0) {
            console.log("❌ Invalid input for hours worked! Please enter a valid positive number.");
            hoursError = true;
        }

        // If both position and hours are invalid, display both error messages
        if (positionError && hoursError) {
            console.log("❌ Both position and hours worked are invalid! Skipping this employee...\n");
            continue; // Skip this employee
        }

        // If only position is invalid, display position error and skip
        if (positionError) {
            console.log("❌ Skipping this employee due to invalid position...\n");
            continue; // Skip this employee
        }

        // If only hours worked is invalid, display hours error and skip
        if (hoursError) {
            console.log("❌ Skipping this employee due to invalid hours worked...\n");
            continue; // Skip this employee
        }

        try {
            // Calculate salary
            let salaryDetails = calculateSalary(position, hoursWorked);

            // Display results
            console.log("\n========= Salary Details =========");
            console.log(`Position      : ${position}`);
            console.log(`Hours Worked  : ${hoursWorked}`);
            console.log(`Hourly Rate   : $${salaryDetails.hourlyRate}`);
            console.log(`Overtime Bonus: $${salaryDetails.overtimeBonus}`);
            console.log(`Total Salary  : $${salaryDetails.totalSalary}`);
            console.log("==================================\n");

        } catch (error) {
            console.log(`❌ Error: ${error.message}`);
            console.log("Skipping to next employee...");
            continue; // Jump to next iteration if an error occurs
        }
    }

    console.log("\n✅ Salary calculation completed for all employees!\n");
}
```

4. Skipping Invalid Entries:

In the main function, when either the position or the hours worked are invalid, I use the continue statement to skip processing that employee. For example, if the position entered is not "Manager", "Developer", or "Intern", the program displays an error message and skips to the

next iteration using continue. Similarly, if the hours worked are invalid, it skips the employee and proceeds to the next one. If both the position and hours are invalid, it shows an error for both and skips that employee.

```
// If both position and hours are invalid, display both error messages
if (positionError && hoursError) {
    console.log("❌ Both position and hours worked are invalid! Skipping this employee...\n");
    continue; // Skip this employee
}

// If only position is invalid, display position error and skip
if (positionError) {
    console.log("❌ Skipping this employee due to invalid position...\n");
    continue; // Skip this employee
}

// If only hours worked is invalid, display hours error and skip
if (hoursError) {
    console.log("❌ Skipping this employee due to invalid hours worked...\n");
    continue; // Skip this employee
}
```

5. Error Handling with try-catch:

To prevent the program from crashing if an error occurs, I wrap the salary calculation in a try-catch block. If an error is thrown (e.g., if an invalid position is provided), it is caught in the catch block, which logs the error message and skips to the next employee. This ensures that the program can continue to process other employees without interruption.

```
    try {
        // Calculate salary
        let salaryDetails = calculateSalary(position, hoursWorked);

        // Display results
        console.log("\n========= Salary Details =========");
        console.log(`Position      : ${position}`);
        console.log(`Hours Worked  : ${hoursWorked}`);
        console.log(`Hourly Rate   : $${salaryDetails.hourlyRate}`);
        console.log(`Overtime Bonus: $${salaryDetails.overtimeBonus}`);
        console.log(`Total Salary  : $${salaryDetails.totalSalary}`);
        console.log("==================================\n");

    } catch (error) {
        console.log(`❌ Error: ${error.message}`);
        console.log("Skipping to next employee...");
        continue; // Jump to next iteration if an error occurs
    }
}

console.log("\n✅ Salary calculation completed for all employees!\n");
```

By combining these techniques, I was able to create a robust and user-friendly employee salary calculator. Invalid entries are handled gracefully, and invalid employees are skipped, allowing the program to continue processing without disruption.

Conclusion

In conclusion, the Employee Salary Calculation Program efficiently handles employee data through the application of several functions. The program assigns hourly rates based on employee positions (Manager, Developer, or Intern) using a switch statement. This method ensures that employees are paid according to their roles while keeping the code easy to read and maintain.

The program also accounts for overtime pay by checking if an employee has worked beyond the standard 160 hours. If so, it calculates an overtime bonus at a 1.5x rate using an if statement, ensuring that employees who work extra hours are fairly compensated.

To process multiple employees, the program uses a for loop. This allows it to efficiently handle a list of employees, iterating through each entry to calculate salaries. The loop structure makes the program scalable and adaptable to larger datasets.

Error handling is integrated with a try-catch block, which captures invalid data such as incorrect positions or negative hours worked. When invalid input is detected, the program displays an error message and skips over the invalid entry, ensuring that the program continues running smoothly without crashing.

Furthermore, the program checks for both invalid positions and hours separately, using continue statements to skip employees with erroneous data. This ensures that only valid entries are processed, while invalid ones are gracefully skipped.

Overall, the combination of position-based salary assignments, overtime calculation, efficient looping, and error handling results in simple user-friendly system for calculating employee salaries. The program ensures accuracy, integrity, and uninterrupted operation, making it an effective tool for salary processing.