**INSPECTION ABOUT FUNTIONALITY**

**INSPECTIONER GROUP 3**

**Test Case of Group #06**

**Functionality: Register employee**

**1. view functionality**

The functionality finds in the use case diagrams

**2. data structure**

The functionality finds in the class diagrams

**3. create your taste case**

{

    "name": "david",

    "lastName": "lainez",

    "idNumber": "0919247024",

    "hireDate": "Jan 15, 2024, 12:00:00 AM",

    "basicSalary": 460.0,

    "overtimeHours": 10.0,

    "absentDays": 0.0,

    "bonuses": 100.0,

    "iessLoans": 3.0,

    "companyLoans": 3.0,

    "fines": 0.0,

"bringOwnFood": true

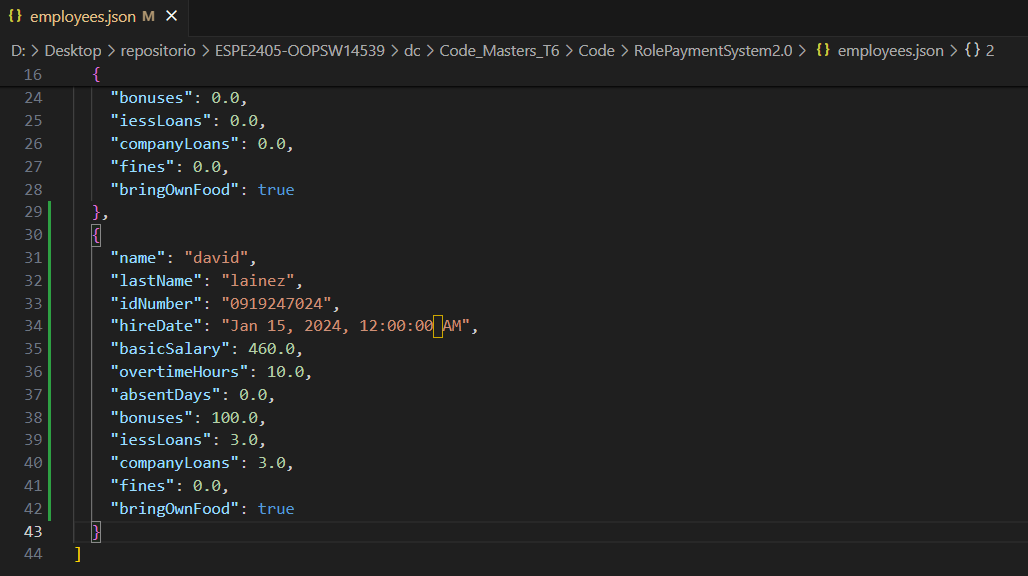
  }

**4. run the system**

Run the program correctly

**5. validate and verify**

It visualizes in the json file correctly.



**Functionality: calculate work hour value**

**1. view functionality**

The functionality finds in the use case diagrams

**2. data structure**

The functionality finds in the class diagrams

**3. create your taste case**

"basicSalary": 460.0,

"overtimeHours": {

          "quantity": 10.0,

          "value": 0.0

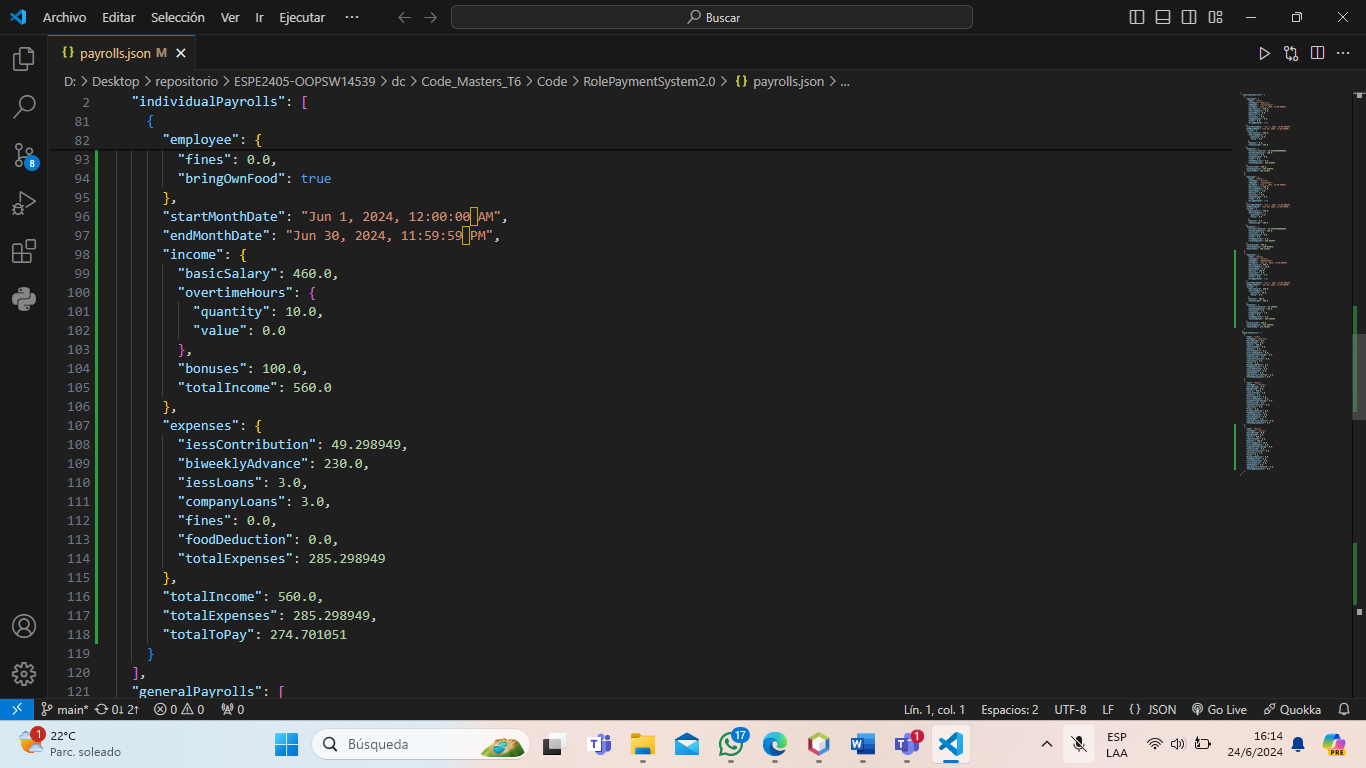
        }

**4. run the system**

Run the program correctly

**5. validate and verify**

It visualizes in the json file correctly.



**Functionality: record hours worked and overtime**

**1. view functionality**

The functionality finds in the use case diagrams

**2. data structure**

The functionality finds in the class diagrams

**3. create your taste case**

"expenses": {

        "iessContribution": 49.298949,

        "biweeklyAdvance": 230.0,

        "iessLoans": 3.0,

        "companyLoans": 3.0,

        "fines": 0.0,

        "foodDeduction": 0.0,

        "totalExpenses": 285.298949

      },

      "totalIncome": 560.0,

      "totalExpenses": 285.298949,

      "totalToPay": 274.701051

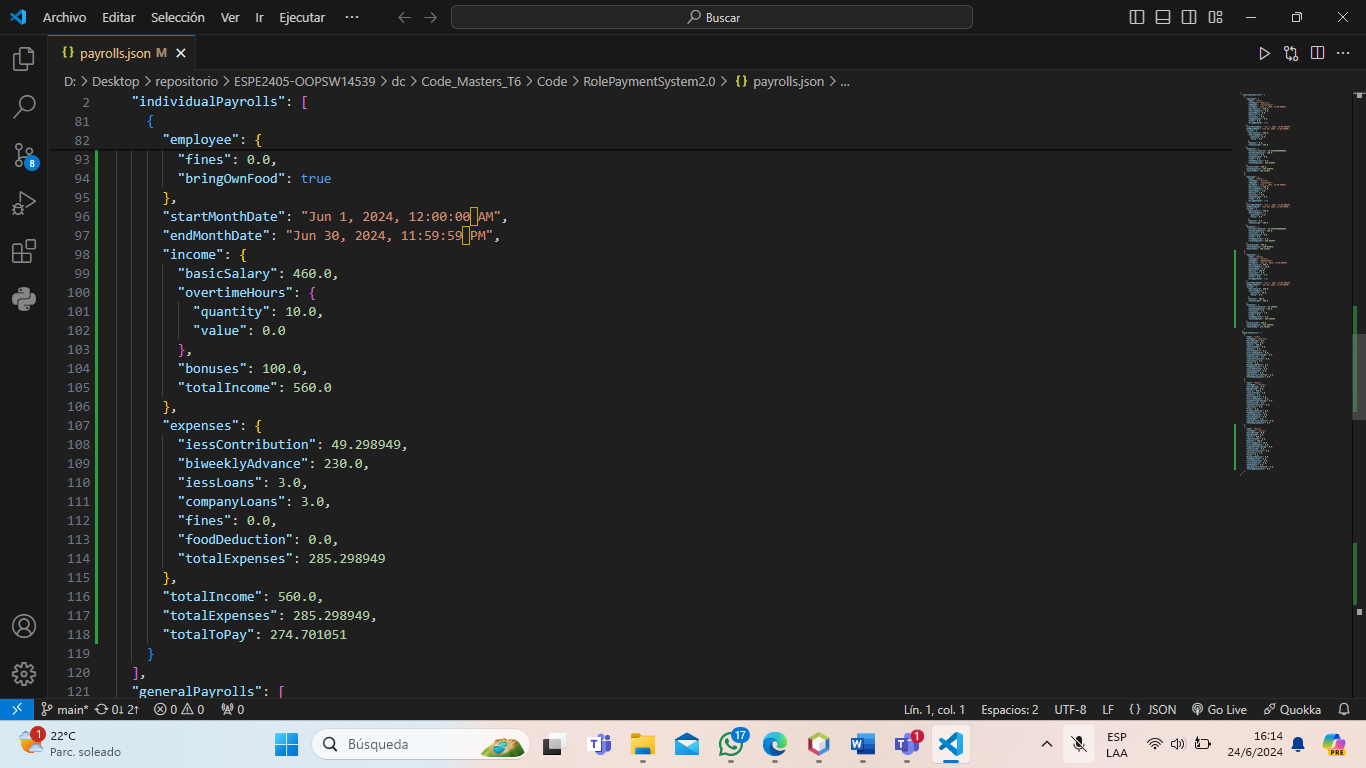
    }

**4. run the system**

Run the program correctly

**5. validate and verify**

It visualizes in the json file correctly.



**Functionality: Reverse Funds**

1. **view Functionality**:

all correct there are no problems with the class in terms of clean code, however there is no consistency with the use case diagram since it is assumed that there is an actor called human talent manager that calculates the same and saves it, however in the code there is no such actor and is calculated separately "alone", on the other hand it does appear in the class diagram.

**Functionality: Calculator**

As the class already indicates that it is going to be used as a calculator, I think it is a good idea to put the word calculate at the beginning of each method.

**Functionality: Overtime Hours Calculation**

The calculateOvertimeHours method assumes that hoursWorked will always be greater than or equal to regularHours. There is no handling of cases where hoursWorked is less, which would result in a negative overtime value.

Division by Zero: In calculateBasicSalary, annualSalary is not checked to be greater than zero to avoid divisions by zero.

As for the test cases, neither of the two functions can be observed since they are "hidden" in the sense that they only work from the back side.

As for pitfalls there are none in both classes and as for the clean code both classes are well written.

There are no errors, as a recommendation to eliminate the headers.

**Functionality: Calculate and Record Reserve Founds**

1. **Functionality**

I found in case diagram, but I don’t find an explicit requirement in IEEE 830

1. **Data Structure**

In the class diagram we found the functionality

1. **Create your test case**

Nombre: Carlos

Apellido: Pérez

Numero de identificación: 1723345678

Fecha de contratación: 15/02/2023

Horas extra: 10

Dias ausentes: 0

Bonificaciones: 5

Préstamos al IESS: 30

Préstamos de la empresa: 10

Multas: 1

Trae su propia comida: false

1. **Json**

The information has been saved correctly

**Functionality: Record Additional Income**

1. **Functionality**

I found in case diagram, but I don’t found an explicit requirement in IEEE 830

1. **Data Structure**

In the class diagram we don’t found the functionality

1. **Create your test case**

Nombre: Andrea

Apellido: Ramírez

Numero de identificación: 1809876543

Fecha de contratación: 10/03/2023

Horas extra: 8

Dias ausentes: 1

Bonificaciones: 7

Préstamos al IESS: 25

Préstamos de la empresa: 15 Multas: 0

Trae su propia comida: true

1. **Json**

The information has been saved correctly

**Functionality: Record Deductions and Expenses**

1. **Functionality**

I found in case diagram, but I don’t found an explicit requirement in IEEE 830

1. **Data Structure**

In the class diagram we found the functionality

1. **Create your test case**

Nombre: Diana

Apellido: Torres

Numero de identificación: 1901234567

Fecha de contratación: 20/05/2023

Horas extra: 7

Dias ausentes: 2

Bonificaciones: 6

Préstamos al IESS: 40

Préstamos de la empresa: 25

Multas: 1

Trae su propia comida: true

1. **Json**

The information has been saved correctly

**Functionality: Expenses**

The Expenses class accurately calculates and encapsulates various expense components for an employee, including IESS contributions, advances, loans, fines, and food deductions.

**Functionality: GeneralPayroll**

The class is overly complex with too many attributes, but it is consistent with the requirements.

**Functionality: Income**

Attribute naming consistency and structure are clear.

**Functionality: IndividualPayroll**

Validating date calculations and ensuring accurate handling of payroll components

**Functionality: OvertimeHours**

The code maintains consistency and provides a clear, straightforward implementation.

**Functionality [**Generate Financial Reports**]:**

We are going to analyze the use case "Generate Financial Reports", the indicated case is found in the requirements document. But it is not found in the class diagram.

**Data Structure:**

In the class diagram we don’t found the functionality.

**Create your test case:**

**Nombre:** Cynthia

**Apellido:** Llumiquinga M

**Numero de identificación:** 1754473773

**Fecha de contratación:** 01/01/2024

**Horas extra:** 5

**Dias ausentes:** 2

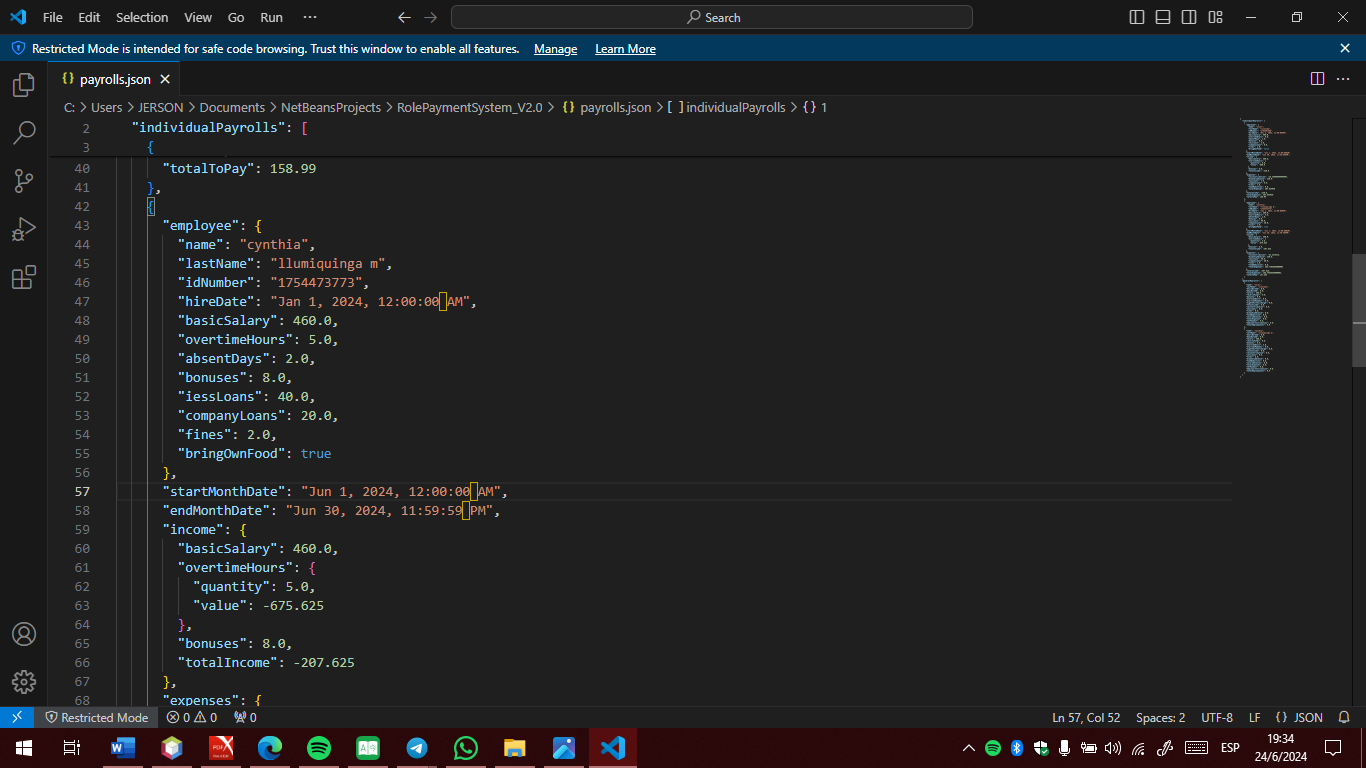
**Bonificaciones:** 8

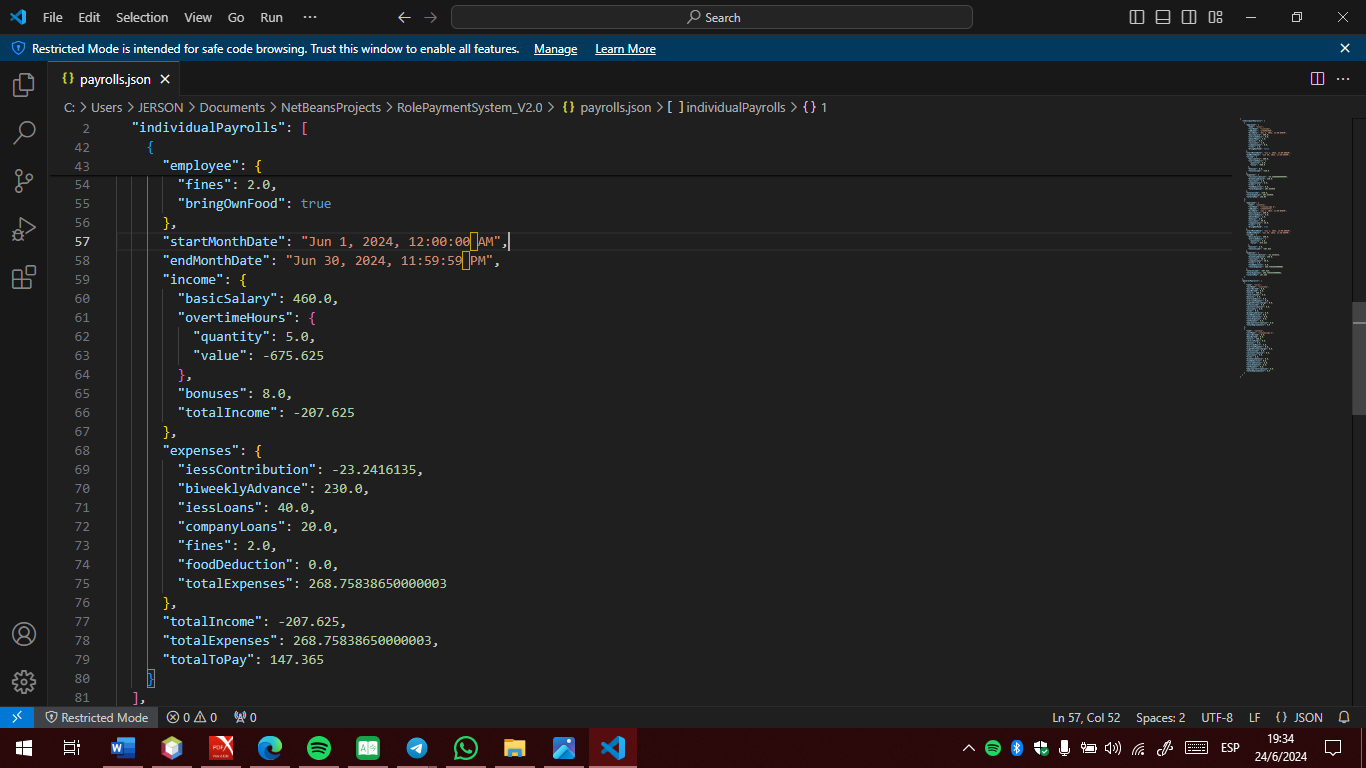
**Préstamos al IESS:** 40

**Préstamos de la empresa:** 20

**Multas:** 2

**Trae su propia comida:** true

The program already create the file JSON with our data



**Class Diagram Consistency**

**MainWindow**

**Functionality [**Update System Parameters**]:**

We are going to analyze the use case " Update System Parameters ", the indicated case is found in the requirements document and in the class diagram in the class, even if it's just "update Employee"

**Data Structure:**

In the class diagram we found the functionality, in the class EmployeeManager.

**Create your test case:**

**Nombre:** Patricia

**Apellido:** Moreno  **Nuevo apellido:** Moreno F.

**Número de identificación:** 1709587230

**Fecha de contratación:** 22/04/2021

**Horas extra:** 4

**Dias ausentes:** 10

**Bonificaciones:** 6

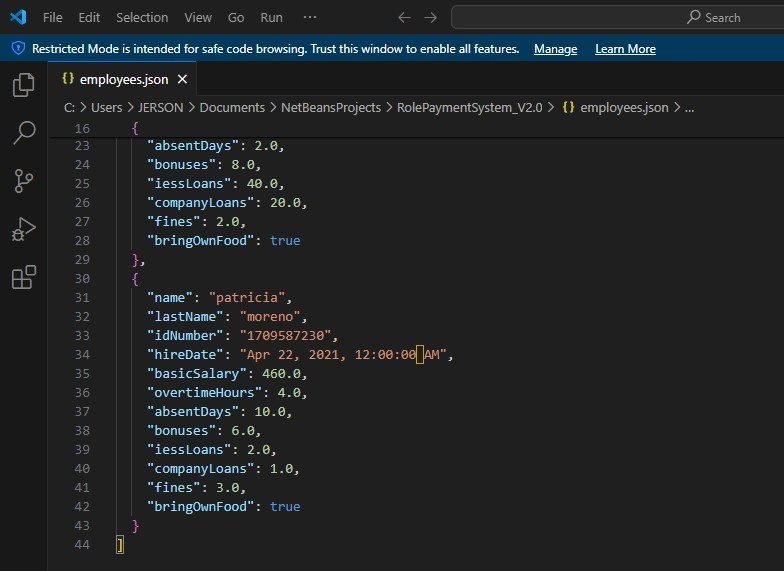
**Préstamos al IESS:** 2

**Préstamos de la empresa:** 1

**Multas:** 3

**Trae su propia comida:** true

The program already create the file JSON with our data



The program updates the parameters, even if they are only the names of the employees.

**Consistencia del diagrama de clases:**

**MainWindow:** The main function exists, but it does not return any value. The correct thing to do would be to enter a return void

**UserInterface:** The functions are in the code correctly, but some have no return.

**Pitfulls and Clean Code:**

The code is quite readable, but there are some aspects in the UserInterface class that can be considered Pifull. For example: Some methods, like addEmployee, are a bit long and could benefit from being split into smaller, more focused methods. The addEmployee function starts at line 56.

Also, we have duplicate code in handling user input (getUserInput, getDoubleInput, etc.) that has some repetition.