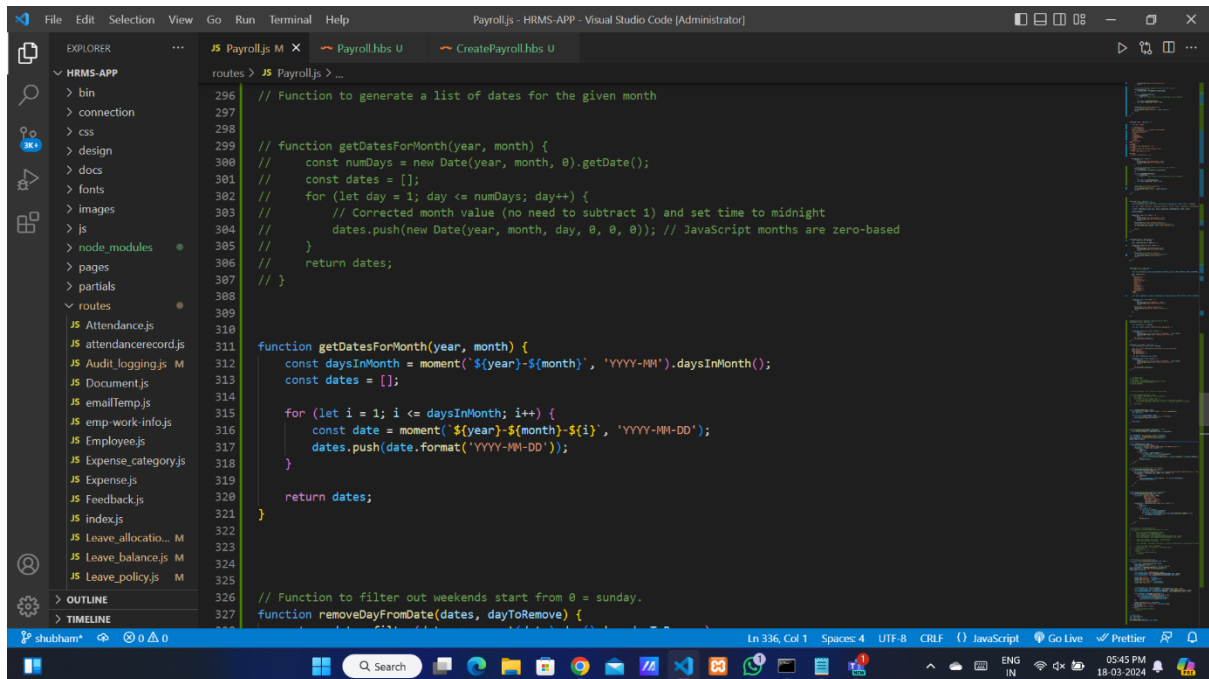


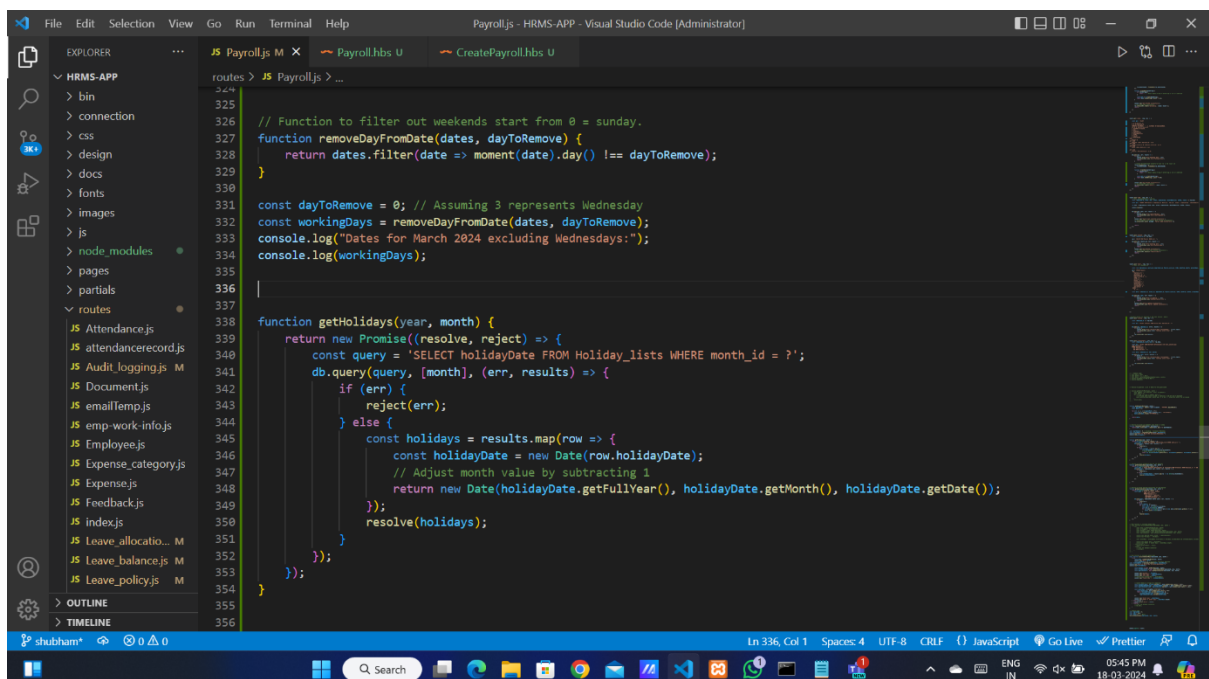
# Payroll Calculation

# function to create total dates of particular month for payroll



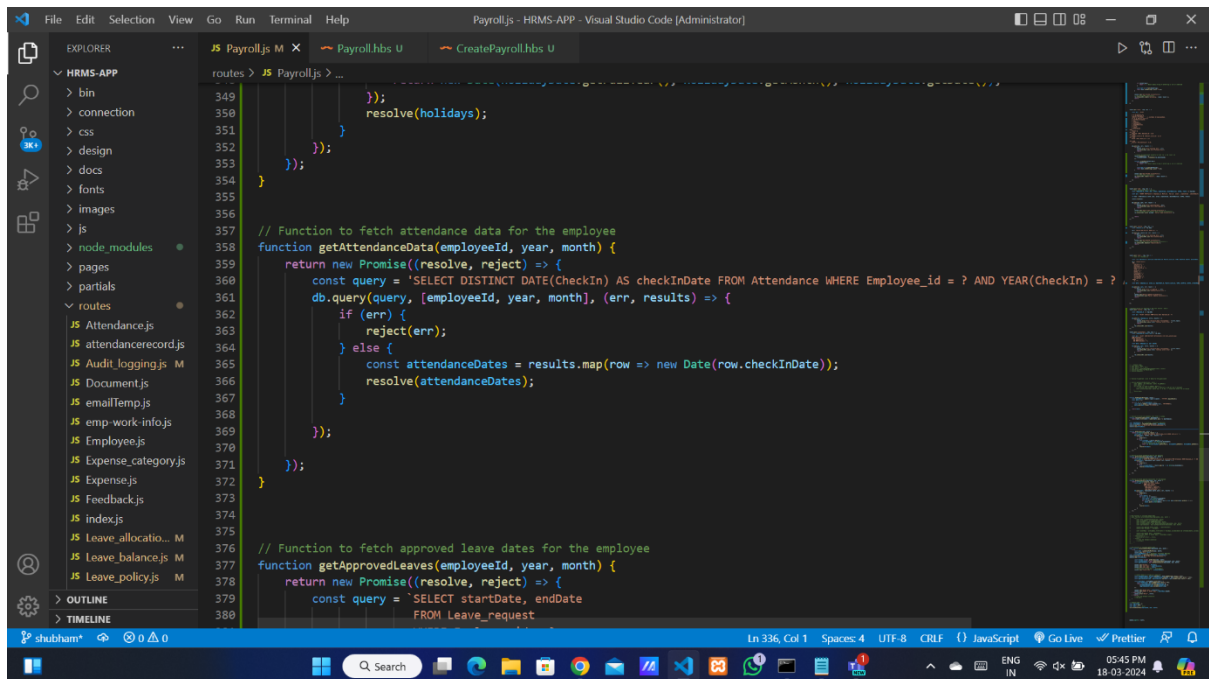
```
296 // Function to generate a list of dates for the given month
297
298
299 // function getDatesForMonth(year, month) {
300 //   const numDays = new Date(year, month, 0).getDate();
301 //   const dates = [];
302 //   for (let day = 1; day <= numDays; day++) {
303 //     // Corrected month value (no need to subtract 1) and set time to midnight
304 //     dates.push(new Date(year, month, day, 0, 0, 0)); // JavaScript months are zero-based
305 //   }
306 //   return dates;
307 // }
308
309
310
311 function getDatesForMonth(year, month) {
312   const daysInMonth = moment(`${year}-${month}`, 'YYYY-MM').daysInMonth();
313   const dates = [];
314
315   for (let i = 1; i <= daysInMonth; i++) {
316     const date = moment(`${year}-${month}-${i}`, 'YYYY-MM-DD');
317     dates.push(date.format('YYYY-MM-DD'));
318   }
319
320   return dates;
321 }
322
323
324 // Function to filter out weekends start from 0 = sunday.
325 function removeDayFromDate(dates, dayToRemove) {
326
```

# Function to remove weekly offs and public holidays from total dates



```
326 // Function to filter out weekends start from 0 = sunday.
327 function removeDayFromDate(dates, dayToRemove) {
328   return dates.filter(date => moment(date).day() !== dayToRemove);
329 }
330
331
332 const dayToRemove = 0; // Assuming 3 represents Wednesday
333 const workingDays = removeDayFromDate(dates, dayToRemove);
334 console.log("Dates for March 2024 excluding Wednesdays:");
335 console.log(workingDays);
336
337
338 function getHolidays(year, month) {
339   return new Promise((resolve, reject) => {
340     const query = 'SELECT holidayDate FROM Holiday_lists WHERE month_id = ?';
341     db.query(query, [month], (err, results) => {
342       if (err) {
343         reject(err);
344       } else {
345         const holidays = results.map(row => {
346           const holidayDate = new Date(row.holidayDate);
347           // Adjust month value by subtracting 1
348           return new Date(holidayDate.getFullYear(), holidayDate.getMonth(), holidayDate.getDate());
349         });
350         resolve(holidays);
351       }
352     });
353   });
354 }
355
356
```

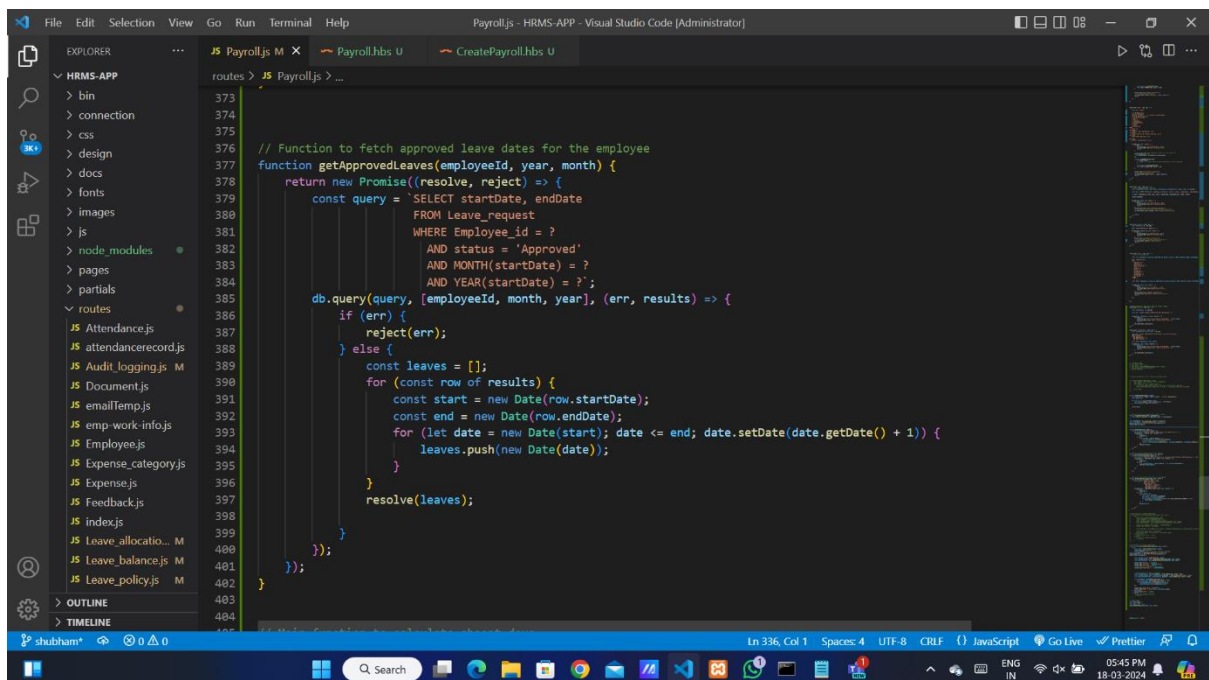
# function to get the total present days in a month  
from attendance table



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `Payroll.js` file. The function `getAttendanceData` is defined, which takes `employeeId`, `year`, and `month` as arguments. It uses a database query to fetch attendance records for the specified employee and month. The function returns a Promise that resolves with an array of dates representing the present days.

```
349  
350  
351  
352  
353  
354  
355  
356  
357 // Function to fetch attendance data for the employee  
358 function getAttendanceData(employeeId, year, month) {  
359   return new Promise((resolve, reject) => {  
360     const query = 'SELECT DISTINCT DATE(CheckIn) AS checkInDate FROM Attendance WHERE Employee_id = ? AND YEAR(CheckIn) = ?';  
361     db.query(query, [employeeId, year, month], (err, results) => {  
362       if (err) {  
363         reject(err);  
364       } else {  
365         const attendanceDates = results.map(row => new Date(row.checkInDate));  
366         resolve(attendanceDates);  
367       }  
368     });  
369   });  
370 }  
371  
372  
373  
374  
375  
376 // Function to fetch approved leave dates for the employee  
377 function getApprovedLeaves(employeeId, year, month) {  
378   return new Promise((resolve, reject) => {  
379     const query = 'SELECT startDate, endDate  
380     FROM Leave_request  
381     WHERE Employee_id = ?  
382     AND status = 'Approved'  
383     AND MONTH(startDate) = ?  
384     AND YEAR(startDate) = ?';  
385     db.query(query, [employeeId, month, year], (err, results) => {  
386       if (err) {  
387         reject(err);  
388       } else {  
389         const leaves = [];  
390         for (const row of results) {  
391           const start = new Date(row.startDate);  
392           const end = new Date(row.endDate);  
393           for (let date = new Date(start); date <= end; date.setDate(date.getDate() + 1)) {  
394             leaves.push(new Date(date));  
395           }  
396         }  
397         resolve(leaves);  
398       }  
399     });  
400   });  
401 }  
402  
403  
404  
405
```

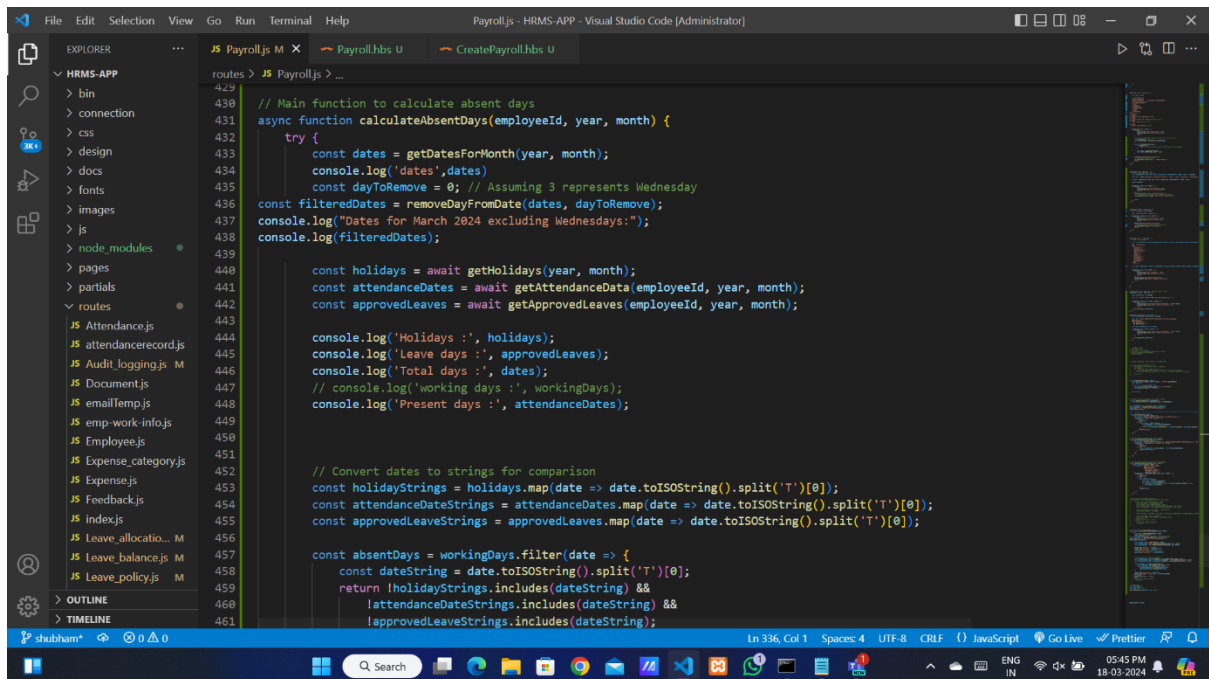
# function to check the approved leaves in that month



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `Payroll.js` file. The function `getApprovedLeaves` is defined, which takes `employeeId`, `year`, and `month` as arguments. It uses a database query to fetch approved leave records for the specified employee and month. The function returns a Promise that resolves with an array of dates representing the approved leaves.

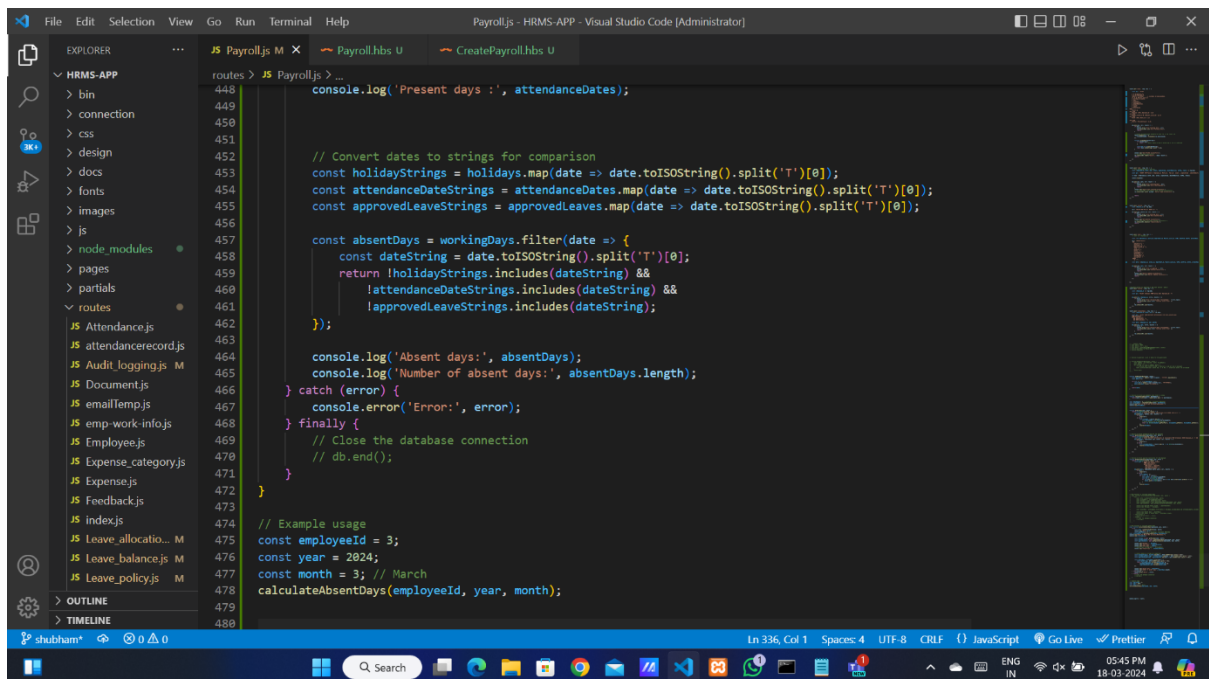
```
373  
374  
375  
376 // Function to fetch approved leave dates for the employee  
377 function getApprovedLeaves(employeeId, year, month) {  
378   return new Promise((resolve, reject) => {  
379     const query = 'SELECT startDate, endDate  
380     FROM Leave_request  
381     WHERE Employee_id = ?  
382     AND status = 'Approved'  
383     AND MONTH(startDate) = ?  
384     AND YEAR(startDate) = ?';  
385     db.query(query, [employeeId, month, year], (err, results) => {  
386       if (err) {  
387         reject(err);  
388       } else {  
389         const leaves = [];  
390         for (const row of results) {  
391           const start = new Date(row.startDate);  
392           const end = new Date(row.endDate);  
393           for (let date = new Date(start); date <= end; date.setDate(date.getDate() + 1)) {  
394             leaves.push(new Date(date));  
395           }  
396         }  
397         resolve(leaves);  
398       }  
399     });  
400   });  
401 }  
402  
403  
404  
405
```

# this function is calculating total absent days after referring attendance , holidays and leaves ; to calculate the salary deduction



This screenshot shows the Visual Studio Code editor with the 'Payroll.js' file open. The function 'calculateAbsentDays' is defined, which takes 'employeeId', 'year', and 'month' as arguments. It uses 'async' and 'await' to fetch data from a database. The function calculates the total number of working days for the given month, then subtracts the number of holidays and approved leaves to determine the total absent days. The function is commented with a note about assuming Wednesday as the day to remove.

```
429 // Main function to calculate absent days
430 async function calculateAbsentDays(employeeId, year, month) {
431   try {
432     const dates = getDatesForMonth(year, month);
433     console.log('dates', dates);
434     const dayToRemove = 0; // Assuming 3 represents Wednesday
435     const filteredDates = removeDayFromDate(dates, dayToRemove);
436     console.log("Dates for March 2024 excluding Wednesdays:");
437     console.log(filteredDates);
438
439     const holidays = await getHolidays(year, month);
440     const attendanceDates = await getAttendanceData(employeeId, year, month);
441     const approvedLeaves = await getApprovedLeaves(employeeId, year, month);
442
443     console.log('Holidays :', holidays);
444     console.log('Leave days :', approvedLeaves);
445     console.log('Total days :', dates);
446     // console.log('Working days :', workingDays);
447     console.log('Present days :', attendanceDates);
448
449     // Convert dates to strings for comparison
450     const holidayStrings = holidays.map(date => date.toISOString().split('T')[0]);
451     const attendanceDateStrings = attendanceDates.map(date => date.toISOString().split('T')[0]);
452     const approvedLeaveStrings = approvedLeaves.map(date => date.toISOString().split('T')[0]);
453
454     const absentDays = workingDays.filter(date => {
455       const dateString = date.toISOString().split('T')[0];
456       return !holidayStrings.includes(dateString) &&
457         !attendanceDateStrings.includes(dateString) &&
458         !approvedLeaveStrings.includes(dateString);
459     });
460   } catch (error) {
461     console.error('Error:', error);
462   } finally {
463     // Close the database connection
464     db.end();
465   }
466 }
```



This screenshot shows the continuation of the 'calculateAbsentDays' function. It includes a catch block for errors and a finally block to close the database connection. The function is then called with example values for employeeId, year, and month.

```
448 console.log('Present days :', attendanceDates);
449
450 // Convert dates to strings for comparison
451 const holidayStrings = holidays.map(date => date.toISOString().split('T')[0]);
452 const attendanceDateStrings = attendanceDates.map(date => date.toISOString().split('T')[0]);
453 const approvedLeaveStrings = approvedLeaves.map(date => date.toISOString().split('T')[0]);
454
455 const absentDays = workingDays.filter(date => {
456   const dateString = date.toISOString().split('T')[0];
457   return !holidayStrings.includes(dateString) &&
458     !attendanceDateStrings.includes(dateString) &&
459     !approvedLeaveStrings.includes(dateString);
460 });
461
462 console.log('Absent days:', absentDays);
463 console.log('Number of absent days:', absentDays.length);
464 } catch (error) {
465   console.error('Error:', error);
466 } finally {
467   // Close the database connection
468   db.end();
469 }
470
471 // Example usage
472 const employeeId = 3;
473 const year = 2024;
474 const month = 3; // March
475 calculateAbsentDays(employeeId, year, month);
476 }
```