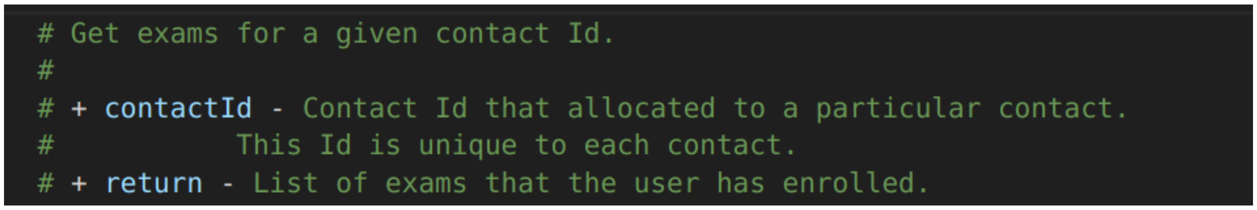
Task 2 - Finding Missing Best Practices and OutDated Code Block

- Repository name: digiops-sales
- Project Name: Certification Portal

1. Inconsistent Use of Punctuation in Code Comments

- **Description:**
  The comment block mixes the use of full stops (periods) at the end of comment lines, especially in parameter and return value descriptions. It is generally considered best practice **not** to use full stops for brief, fragment-style documentation entries (such as `+ contactId - ...`). Using periods inconsistently can reduce readability and introduce a lack of uniformity across the codebase.

- **Recommendation:**
  Omit full stops in short, non-sentence descriptions for parameters and return values. Reserve punctuation for full sentences explaining logic or behavior. Ensure consistency across all comment blocks.



```
# Get exams for a given contact Id.
#
# + contactId - Contact Id that allocated to a particular contact.
#               This Id is unique to each contact.
# + return - List of exams that the user has enrolled.
```

2. Unnecessary Space Before Opening Curly Brace

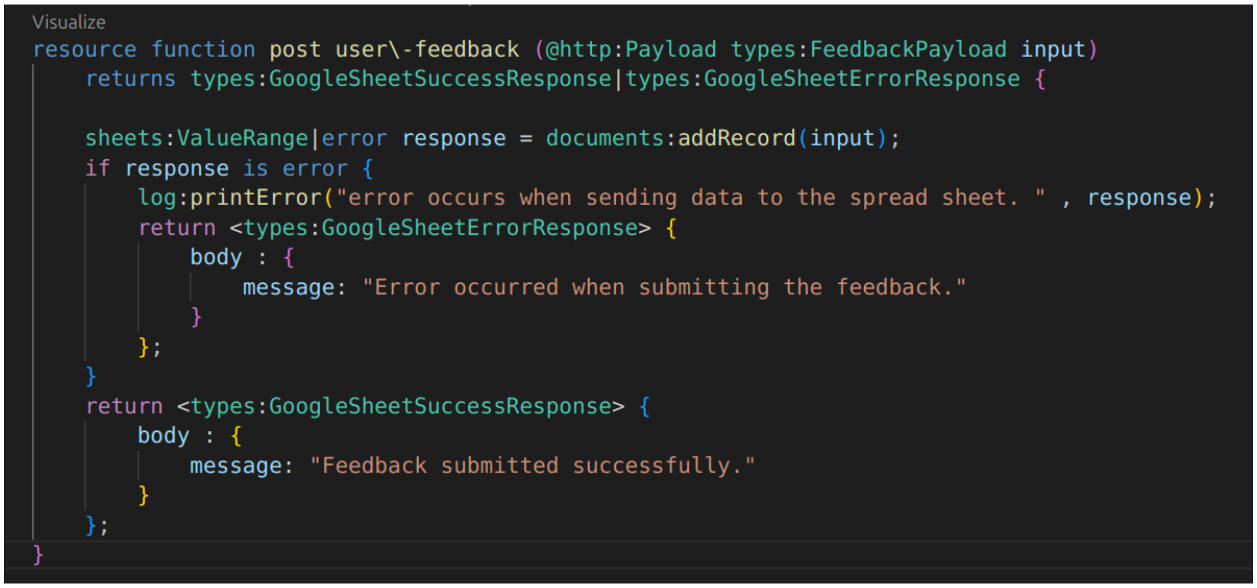
- **Description:**
  There is an unnecessary space before the opening curly brace { following the return type declaration.
  This goes against standard formatting conventions. Consistent spacing improves code readability and aligns with common best practices in most programming languages, including Ballerina.

Unset
```
returns
types:GoogleSheetSuccessResponse|types:GoogleSheetErrorR
esponse {
```

- **Recommendation:**

  Remove the space before {

Unset
```
returns
types:GoogleSheetSuccessResponse|types:GoogleSheetErrorR
esponse{
```



Visualize
```
resource function post user\-feedback (@http:Payload types:FeedbackPayload input)
    returns types:GoogleSheetSuccessResponse|types:GoogleSheetErrorResponse {

    sheets:ValueRange|error response = documents:addRecord(input);
    if response is error {
        log:printError("error occurs when sending data to the spread sheet. " , response);
        return <types:GoogleSheetErrorResponse> {
            body : {
                message: "Error occurred when submitting the feedback."
            }
        };
    }
    return <types:GoogleSheetSuccessResponse> {
        body : {
            message: "Feedback submitted successfully."
        }
    };
}
```

3. Unnecessary Double Space in Log Message

- **Description:**
  In the following line:

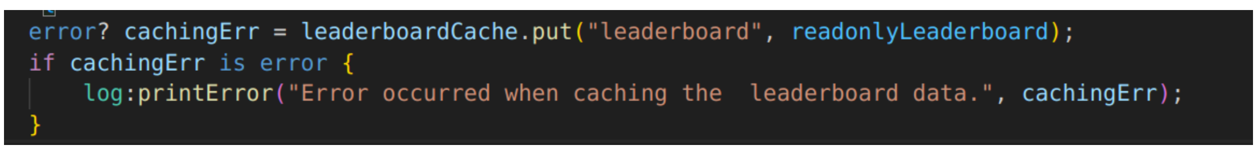
```
Unset
    log:printError("Error occurred when caching the  leaderboard
    data.", cachingErr);
```

There is an **unintentional double space** between `"the"` and `"leaderboard"`. While it may seem minor, such inconsistencies can reduce the readability and professionalism of logs, especially when analyzing them in production environments or debugging tools.

- **Recommendation:**
  Replace the double space with a single space to maintain clean and consistent formatting:

```
Unset
    log:printError("Error occurred when caching the leaderboard
    data.", cachingErr);
```

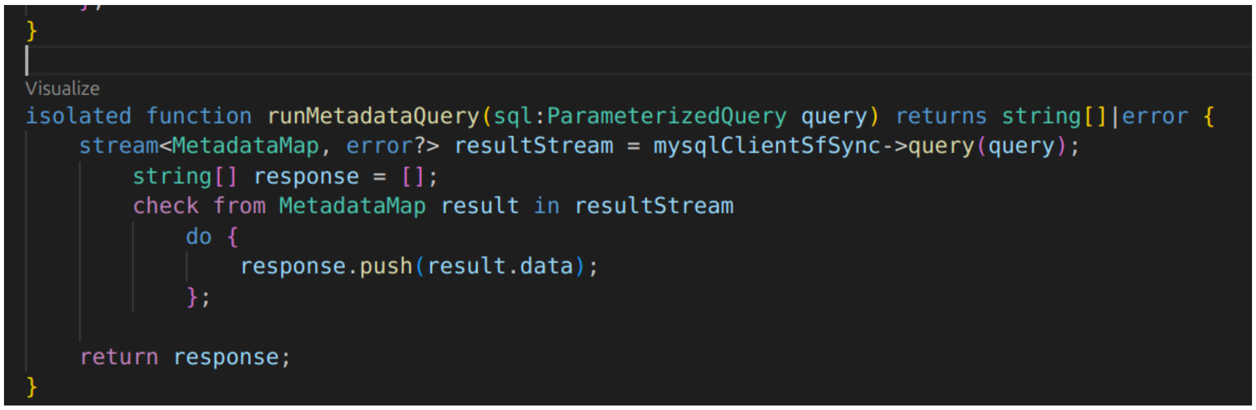
Maintaining proper spacing improves the clarity of messages and aligns with standard best practices for clean code and logging.



```
error? cachingErr = leaderboardCache.put("leaderboard", readonlyLeaderboard);
if cachingErr is error {
    log:printError("Error occurred when caching the  leaderboard data.", cachingErr);
}
```

4. Missing Descriptive Comment for `runMetadataQuery` Function

- **Description:**
  The `runMetadataQuery` function lacks a descriptive comment explaining its purpose, parameters, and return type. Without documentation, it's unclear what the function does, what kind of query it expects, and what `response` contains. This can lead to misunderstandings and reduce code maintainability.
- **Recommendation:**
  Add a function-level comment that briefly explains:



```
}

Visualize
isolated function runMetadataQuery(sql:ParameterizedQuery query) returns string[]|error {
    stream<MetadataMap, error?> resultStream = mysqlClientSfSync->query(query);
        string[] response = [];
        check from MetadataMap result in resultStream
            do {
                response.push(result.data);
            };

    return response;
}
```

5. Snake Case Used Instead of Camel Case

- **Description:**
  In the following map definition:

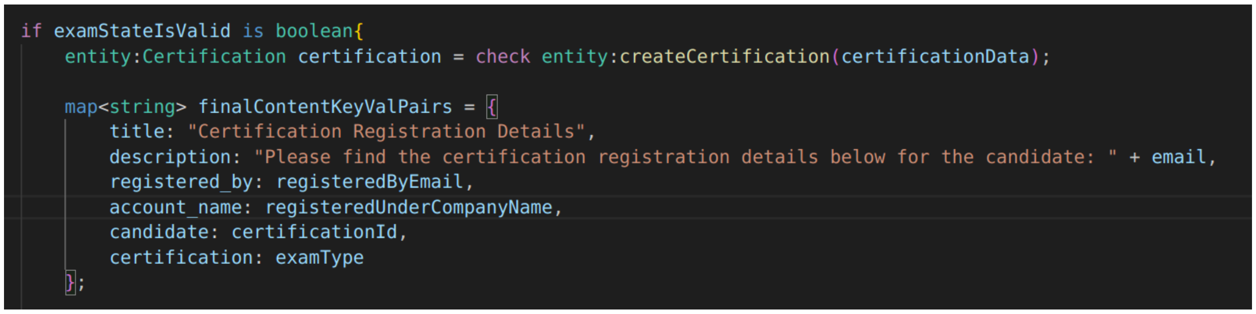
```
Unset
map<string> finalContentKeyValPairs = {
    registered_by: registeredByEmail,
    account_name: registeredUnderCompanyName,
    ...
};
```

The keys `registered_by` and `account_name` use **snake_case**, which goes against the standard **camelCase (lowerCamelCase)** naming convention typically followed in Ballerina and most modern languages. This inconsistency can lead to confusion and reduce code readability, especially when naming patterns vary across the codebase.

- **Recommendation:**
  Update the key names to use **lowerCamelCase** for consistency with Ballerina coding standards:

```
Unset
map<string> finalContentKeyValPairs = {
    registeredBy: registeredByEmail,
    accountName: registeredUnderCompanyName,
    ...
};
```

Consistent use of camelCase improves readability, aligns with best practices, and helps maintain a professional and clean codebase. If snake_case is required by an external API spec (e.g., a specific JSON format), consider using serialization annotations instead of altering variable naming.



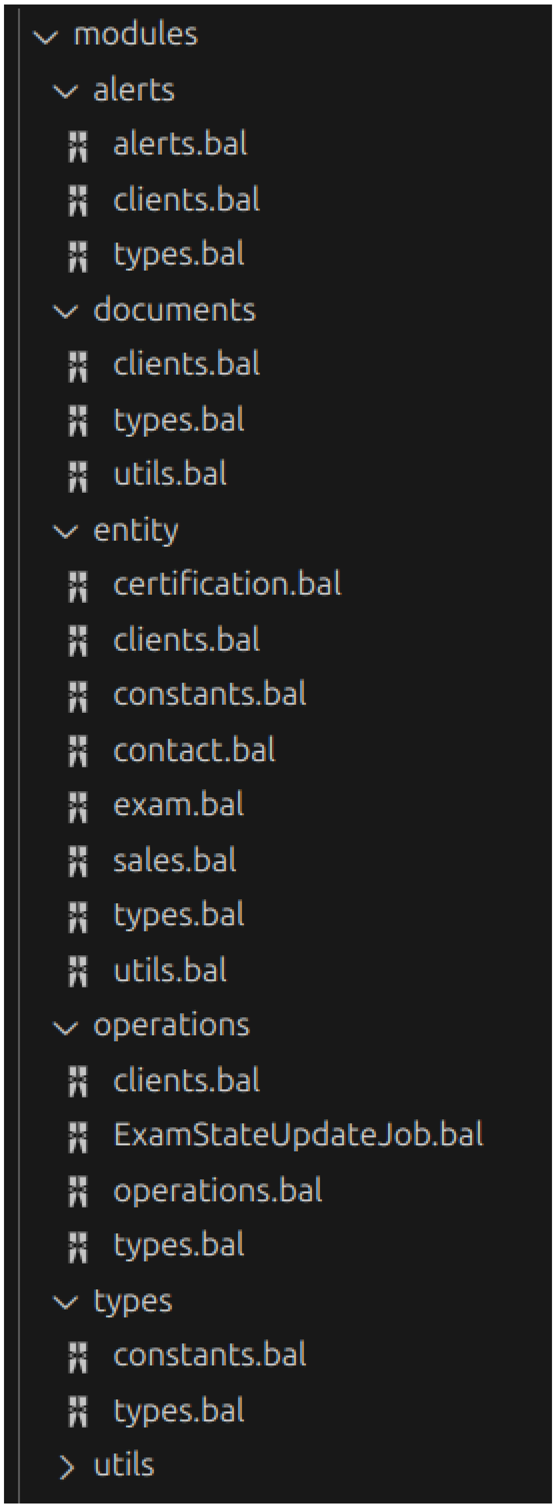
```
if examStateIsValid is boolean{
    entity:Certification certification = check entity:createCertification(certificationData);

    map<string> finalContentKeyValPairs = {
        title: "Certification Registration Details",
        description: "Please find the certification registration details below for the candidate: " + email,
        registered_by: registeredByEmail,
        account_name: registeredUnderCompanyName,
        candidate: certificationId,
        certification: examType
    };
```

6. Missing `Module.md` Files in Module Directories

- **Description:**
  The folders inside the `modules/` directory (such as `alerts`, `documents`, `entity`, `operations`, `types`, and `utils`) are missing corresponding `Module.md` files. In Ballerina projects, it's a best practice to include a `Module.md` file in each module folder to provide clear documentation about the purpose and usage of that module.

  The absence of `Module.md` files reduces clarity, especially when the codebase grows or is used by multiple developers. It also goes against standard Ballerina project documentation guidelines.

- **Recommendation:**
  Add a `Module.md` file to each module directory under `modules/`. At a minimum, each file should include:



```
modules
  alerts
    alerts.bal
    clients.bal
    types.bal
  documents
    clients.bal
    types.bal
    utils.bal
  entity
    certification.bal
    clients.bal
    constants.bal
    contact.bal
    exam.bal
    sales.bal
    types.bal
    utils.bal
  operations
    clients.bal
    ExamStateUpdateJob.bal
    operations.bal
    types.bal
  types
    constants.bal
    types.bal
  utils
```

7. Singular Resource Name in `post user-feedback`

- **Description:**
  The resource function is defined with a singular name in the path:

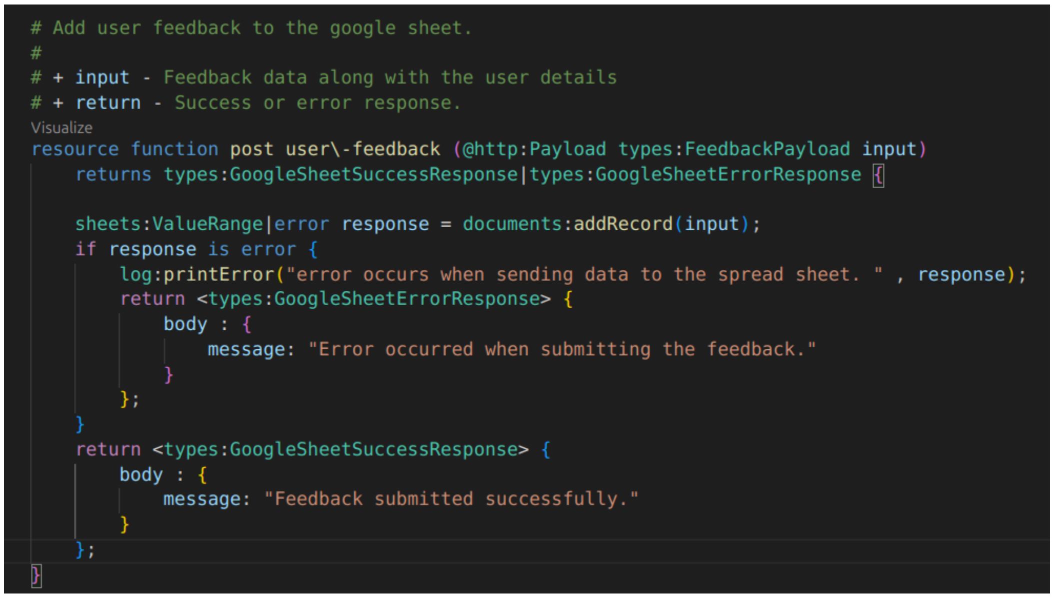
```
Unset
resource function post user\-feedback (@http:Payload
types:FeedbackPayload input)
```

Using `"user\-feedback"` (singular) as the resource noun is not aligned with RESTful API design or Ballerina's resource naming conventions. The singular form can limit semantic clarity, especially when you're dealing with collections or multiple items in future endpoints.

- Recommendation:

  Rename the resource path to use the **plural form** — `user\-feedbacks`:

```
Unset
resource function post user\-feedbacks(@http:Payload
types:FeedbackPayload input)
```



```
# Add user feedback to the google sheet.
#
# + input - Feedback data along with the user details
# + return - Success or error response.
Visualize
resource function post user\-feedback (@http:Payload types:FeedbackPayload input)
    returns types:GoogleSheetSuccessResponse|types:GoogleSheetErrorResponse {

    sheets:ValueRange|error response = documents:addRecord(input);
    if response is error {
        log:printError("error occurs when sending data to the spread sheet. " , response);
        return <types:GoogleSheetErrorResponse> {
            body : {
                message: "Error occurred when submitting the feedback."
            }
        };
    }
    return <types:GoogleSheetSuccessResponse> {
        body : {
            message: "Feedback submitted successfully."
        }
    };
}
```