API Implementation Exercise

In this exercise, you will implement an API in Golang that enables users to send requests for reading and writing alert data to a data storage system.

Guidelines

- You can chose any storage system; a relational database, a file system, or even an in-memory data structure.
- You can use your favorite Integrated Development Environment (IDE), Framework, or the standard libraries provided by your chosen programming language.
- For bonus points, consider including unit tests to ensure the reliability and correctness of your API.
- Please ensure that you upload your completed solutions to GitHub and share the repository link with us. This will allow us to access and review your work effectively.

Academic Integrity Notice:

Candidates are expected to complete this assignment independently, without assistance from external sources, including the internet and AI-powered tools like ChatGPT. We value academic integrity and will assess your work based on your understanding and abilities.

Please be aware that we have mechanisms in place to identify instances of plagiarism or unauthorized assistance. We will thoroughly evaluate your submission to ensure its originality and adherence to our academic integrity guidelines. Any violations may be detected during the interview process, which may impact your candidacy.

Please proceed with honesty and professionalism while completing this assignment.

Write Alerts

Users should be able to send requests to this API to write alert data to the chosen data storage.

Write Request

HTTP Method: POST

Endpoint: /alerts

Request Body:

```
{
   "alert_id": "b950482e9911ec7e41f7ca5e5d9a424f",
   "service_id": "my_test_service_id",
   "service_name": "my_test_service",
   "model": "my_test_model",
   "alert_type": "anomaly",
   "alert_ts": "1695644160",
   "severity": "warning",
   "team_slack": "slack_ch"
}
```

Write Response

Success

HTTP Status Code: 200 OK

Response Body:

```
{
   "alert_id": "b950482e9911ec7e41f7ca5e5d9a424f",
   "error": ""
}
```

Error

HTTP Status Code: 500 Internal Server Error

Response Body:

```
{
   "alert_id": "b950482e9911ec7e41f7ca5e5d9a424f",
   "error": "<error details>"
}
```

Read Alerts

Users should be able to query alerts using the `service_id` and specifying a time period defined by `start_ts` and `end_ts`.

Read Request

HTTP Method: GET

Endpoint: /alerts

Query Parameters:

- `service_id`: The identifier of the service for which alerts are requested.
- `start_ts`: The starting timestamp epoch of the time period.
- `end_ts`: The ending timestamp epoch of the time period.

Example: `/alerts?service_id=my_test_service_id&start_ts=1695643160&end_ts=1695644360`

Read Response

Success

HTTP Status Code: 200 OK

Response Body:

```
{
   "service_id" : "my_test_service_id"
   "service_name": "my_test_service",
   "alerts" : [
      {
      "alert_id": "b950482e9911ec7e41f7ca5e5d9a424f",
      "model": "my_test_model",
      "alert_type": "anomaly",
      "alert_ts": "1695644060",
      "severity": "warning",
      "team_slack": "slack_ch"
      },
      {
         "alert_id": "b950482e9911ecsdfs41f75e5d9az23cv",
         "model": "my_test_model",
         "alert_type": "anomaly",
         "alert_ts": "1695644160",
         "severity": "warning",
         "team_slack": "slack_ch"
      },
   ]
}
```

Error

HTTP Status Code: Appropriate HTTP error status (e.g., 400 Bad Request, 404 Not Found, 500 Internal Server Error)

Response Body:

```
{
   "alert_id": "b950482e9911ec7e41f7ca5e5d9a424f",
   "error": "<error details>"
}
```