Golang Country Search API - Technical Assessment

Overview

Build a REST API service that provides country information by leveraging the REST Countries API (**https://restcountries.com/**). The service should include custom caching logic and unit testing.

Core Requirements

1. REST API Endpoint

- Create a single endpoint: `GET /api/countries/search`
- Query parameter: `name` (string) for country search
- Response format (JSON):

```
{
    "name": "string",
    "capital": "string",
    "currency": "string",
    "population": number
}
```

- Example-

```
# Request
curl -X GET http://localhost:8000/api/countries/search?name=India

# Response
{
    "name": "India",
    "capital": "New Delhi",
    "currency": "₹",
    "population": 1380004385
}
```

2. Working

- When the user calls the search API (e.g. `/api/countries/search?name=India`), first check if data exists in cache.
- If data exists in cache, then retrieve data from cache and send the response to the user.

- If data doesn't exist in cache, call the 3rd party API, store the result in cache, and serve the response to the user.

3. Custom Cache Implementation

- Implement an in-memory cache from scratch (no external libraries)
- Cache interface should support:
    - Get operation
    - Set operation
    - Thread-safe operations

4. Technical Requirements

Implementation Details:

- Use Go modules for dependency management
- Implement proper error handling and logging
- Use context for handling timeouts
- Implement graceful shutdown

Required Components:

HTTP Client

- Implement a custom HTTP client for REST Countries API
- Handle errors and timeouts
- Parse and validate responses

Cache Implementation

```go
type Cache interface {
    Get(key string) (interface{}, bool)
    Set(key string, value interface{})
}
```

Service Layer

- Business logic for country data processing
- Cache interaction logic
- Error handling and validation

5. Testing Requirements

Unit Tests

- Cache implementation
- HTTP client
- Service layer
- API handlers

Race Condition Tests:

- Concurrent cache access
- Multiple simultaneous API requests

Evaluation Criteria

Code Quality

- Clean, readable code
- Proper error handling
- Documentation
- Go idioms and patterns
- Project structure

Testing

- Test quality
- Test coverage
- Race condition handling

Documentation

- API documentation
- Setup instructions
- Code comments

Time Allocation

- Maximum time: 2 Days

Submission

- Create a GitHub repository and push your work on it
- Share the repository link to begin the evaluation