

Assignment 3: Customizing the Go API Codebase to Our Application

Environment Setup

Item	Details
Operating System	Linux
Programming Lang	Go
Git Repo	https://github.com/Muditha-Kumara/Go/tree/task2
Original Project Name	Vehicle Behavior Monitoring System
Git Repo	https://github.com/Muditha-Kumara/IntelligentDevices

Project Structure:

```
API 0.1/
  cmd/api/main.go
  internal/api/handlers/
  internal/api/middleware/
  internal/api/repository/
  internal/api/server/
  internal/api/service/
```



Introduction

This report documents the process of customizing the Go backend codebase for intelligent devices for Vehicle Behavior Monitoring Project, as per Assignment 3. The main focus was to redesign the data entity, update the database schema, adjust handlers, add validation, and implement logic to handle rapid POST requests.

1. Codebase Exploration

The project uses a Handler-Service-Repository-Model pattern:

- **Handler Layer:** HTTP request/response logic (`internal/api/handlers/data/`)
- **Service Layer:** Business logic and validation (`internal/api/service/data/`)
- **Repository Layer:** Database access (`internal/api/repository/DAL/SQLite/`)
- **Model Layer:** Data structures (`internal/api/repository/models/`)

2. Defining the Device Data Entity

New Data Entity:

```
type Data struct {
    DeviceID    string
    VehicalID   string
    Data         string
    AlertType    string
    DateTime     string
    Location     string
}
```

- `vehical_id` (note: spelling as per your structure) and `alert_type` are new fields.
- `data` is a string for device sensors data.
- `location` and `date_time` are included for context.

The screenshot shows a Visual Studio Code interface with the following components:

- Code Editor:** Displays a Go file named `getbyid.go`. The code implements a `GetByIDHandler` function that handles GET requests for vehicle data. It includes logic to validate vehicle IDs and handle multiple POST requests from the same device.
- Terminal:** Shows the output of a curl command:


```
mudit@Lap-:~/Go$ curl -X GET http://127.0.0.1:8080/data/veh1 -i -u admin123:Testing@123 -H "Content-Type: application/json"
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Date: Sun, 14 Dec 2025 16:25:58 GMT
Content-Length: 137
```
- SQLite Database Viewer:** Shows a table named `data` with three rows:

device_id	vehical_id	data	alert_type	date_time
dev1	veh1	some data	warning	2025-12-14T12:00:00Z
dev2	veh2	some data	critical	2025-12-14T12:00:00Z
- Bottom Status Bar:** Shows the current commit status, file size, and other development metrics.

3. Implementing Custom Data Entity and Handlers

- The new struct was added to the models.
- The SQLite table schema was updated to match the new fields.
- All CRUD handlers in `internal/api/handlers/data/` were updated to use the new structure.
- The POST handler was modified: if two POST requests are received from the same device in less than 1 minute, the API responds with a "warning" message instead of "ok" message. Next the device should configure according to post reply.

Example POST Payload:

```
{
  "device_id": "dev3",
  "vehical_id": "veh3",
  "data": "some data",
  "alert_type": "warning",
  "date_time": "2025-12-14T12:00:00Z",
  "location": "lab"
}
```

The screenshot shows a Visual Studio Code interface with the following components:

- EXPLORER**: Shows the project structure with files like `2nd_Assignment`, `API 0.1`, `cmd/api`, `main.go`, `production.db`, `internal/api`, `handlers/data`, and test files.
- CODE EDITOR**: Displays `post.go` containing Go code for a POST handler. It includes comments explaining the validation logic for device_id, vehical_id, data, and date_time fields, and handles user errors for invalid JSON payloads.
- TERMINAL**: Shows command-line output from curl requests to the API endpoint, demonstrating successful responses for different device IDs and error responses for invalid input.
- TEST RESULTS**: Shows results of tests run on the code.
- DEBUG CONSOLE**: Shows the current state of the debugger.
- SPELL CHECKER**: Shows the status of spell checking.
- OUTLINE**: Shows the outline of the code.
- TIMELINE**: Shows the timeline of changes.
- PROJECT COMPONENTS**: Shows the components of the project.
- GO**: Shows Go-related metrics.
- task3* 18 Git Graph**: Shows a git graph.
- DATA VIEWS**: Shows a table named `data` with columns `device_id`, `vehical_id`, `data`, `alert_type`, and `date_time`. It contains two rows of data: `dev1, veh1, some data, warning, 2025-12-14T12:00:00Z` and `dev2, veh2, some data, critical, 2025-12-14T12:00:00Z`.
- TEST RESULTS**: Shows the results of unit tests.

4. Setting Up and Customizing Validators

- Validators were implemented to check required fields (`device_id`, `vehical_id`, `data`, `date_time`).
- Additional checks: `alert_type` must be one of the allowed values (e.g., "info", "warning", "critical").
- Validation is performed before any database operation in the service layer.

5. Integrating Device with Backend

- Devices send data to the backend using the new POST endpoint.
- The backend responds with a message after process.

6. Advanced: Adding Intelligence

- The POST handler logic prevents rapid submissions (less than 1 minute apart) from the same device, send warn message to device and inform to driver.

The screenshot shows a Visual Studio Code interface with multiple panes. The left pane is the Explorer, showing a file tree with various Go files like `get.go`, `getbyid.go`, and `put.go`. The center-left pane is the Code Editor with a Go file containing code for handling HTTP requests. The center-right pane is the SQLite Database Explorer showing a table named 'data' with columns: device_id, vehical_id, data, alert_type, date_time, and location. The bottom right pane is the Terminal, showing a command-line session where a POST request is made to a local API endpoint, followed by two GET requests to verify the data. The status bar at the bottom indicates the command was run 23 hours ago.

```

    import (
        "context"
        "encoding/json"
        "log"
        "net/http"
        "strconv"
        "time"
    )

    // * The GET method retrieves all resources identified by a URI *
    // * e.g., > GET http://127.0.0.1:8880/data -i u admin123:Testing@123_H="Content-Type: application/json"
    func GetHandler(w http.ResponseWriter, *http.Request, logger *log.Logger, ds service.DataService) {
        page, err := strconv.Atoi(r.URL.Query().Get("page"))
        if err != nil {
            if err == strconv.NumError {
                // * There was no page specified, so we default to 0 *
                page = 0
            } else {
                // * Invalid page specified, return a 400 status code *
                w.WriteHeader(http.StatusBadRequest)
                w.Write([]byte(`{"error": "Invalid page specified."}`))
                return
            }
        }
        ctx, cancel := context.WithTimeout(r.Context(), 2*time.Second)
        defer cancel()
        ...
    }

    muditha@lap:~/Go$ curl -X POST http://127.0.0.1:8880/data -i u admin123:Testing@123_H="Content-Type: application/json" -d {"device_id": "dev2", "vehical_id": "veh2", "data": "some data", "alert_type": "critical", "date_time": "2025-12-14T12:50:00Z", "location": "lab"}
    HTTP/1.1 201 Created
    Access-Control-Allow-Origin: *
    Content-Type: application/json
    Date: Sun, 14 Dec 2025 15:51:24 GMT
    Content-Length: 136
    {"device_id": "dev2", "vehical_id": "veh2", "data": "some data", "alert_type": "critical", "date_time": "2025-12-14T12:50:00Z", "location": "lab"}
    muditha@lap:~/Go$ curl -X GET http://127.0.0.1:8880/data -i u admin123:Testing@123_H="Content-Type: application/json"
    HTTP/1.1 200 OK
    Access-Control-Allow-Origin: *
    Content-Type: application/json
    Date: Sun, 14 Dec 2025 15:53:53 GMT
    Content-Length: 273
    {"device_id": "dev2", "vehical_id": "veh2", "data": "some data", "alert_type": "critical", "date_time": "2025-12-14T12:00:00Z", "location": "lab"}, {"device_id": "dev2", "vehical_id": "veh2", "data": "some data", "alert_type": "warning", "date_time": "2025-12-14T12:50:00Z", "location": "lab"}]
    muditha@lap:~/Go$ 

```

7. Testing and Debugging

- Unit tests were updated for the new data structure and logic.
- Manual API testing was performed using Thunder Client/Postman.
- The rapid POST logic was tested by sending two requests within 1 minute and confirming the correct response.

The screenshot shows a Visual Studio Code interface with the terminal window active. The terminal output shows a Go test run for a package named 'internal/api/handlers/data'. The test results indicate several successful runs and one failure due to a missing device ID. The status bar at the bottom indicates the command was run 1 hour and 22 minutes ago.

```

    muditha@lap:~/Go/2nd_Assignment> go test -v ./...
    === RUN TestDeleteInvalidID
    --- PASS: TestDeleteInvalidID (0.00s)
    === RUN TestDeleteError
    2025/12/14 19:23:23 Could not delete data: Error deleting data. dev1
    --- PASS: TestDeleteError (0.00s)
    === RUN TestDeleteNotFound
    --- PASS: TestDeleteNotFound (0.00s)
    === RUN TestDeleteSuccessful
    --- PASS: TestDeleteSuccessful (0.00s)
    === RUN TestGetHandlerSuccessful
    --- PASS: TestGetHandlerSuccessful (0.00s)
    === RUN TestGetHandlerNotFound
    --- PASS: TestGetHandlerNotFound (0.00s)
    === RUN TestGetHandlerError
    2025/12/14 19:23:23 Could not get data: Error reading data. []
    --- PASS: TestGetHandlerError (0.00s)
    === RUN Test GetByIdInvalidID
    --- PASS: Test GetByIdInvalidID (0.00s)
    === RUN Test GetByIdInternalError
    2025/12/14 19:23:23 Could not read by vehical_id: Error reading data. 1
    --- PASS: Test GetByIdInternalError (0.00s)
    === RUN Test GetByIdNotFound
    --- PASS: Test GetByIdNotFound (0.00s)
    === RUN Test GetByIdSuccessful
    --- PASS: Test GetByIdSuccessful (0.00s)
    === RUN Test Options
    --- PASS: Test Options (0.00s)
    === RUN TestPostInvalidRequestBody
    --- PASS: TestPostInvalidRequestBody (0.00s)
    === RUN TestPutHandlerError
    --- PASS: TestPutHandlerError (0.00s)
    === RUN TestPutDataNotFound
    --- PASS: TestPutDataNotFound (0.00s)
    === RUN TestPutHandlerSuccess
    --- PASS: TestPutHandlerSuccess (0.00s)
    PASS
    ok   goapi/internal/api/handlers/data      0.006s

```

8. Documentation and Reflection

Customization Steps

- Redesigned the data entity and updated all layers (model, handler, service, repository).
- Updated the database schema and migration logic.
- Implemented new validation and anti-flooding logic in the POST handler.
- Updated and ran all tests.

Reflection

This assignment deepened my understanding of Go backend architecture, especially in customizing data flows and enforcing business rules. The main challenge was ensuring the rapid POST detection logic worked reliably and updating all layers to match the new data structure. I improved my skills in RESTful API design, validation, and test-driven development. For future improvements, I would consider adding rate-limiting middleware and more granular error handling.



Conclusion

- The Data entity and all related layers were successfully customized.
- The database schema and handlers were updated.
- Validators and Warn logic were implemented.
- All changes were tested and verified.