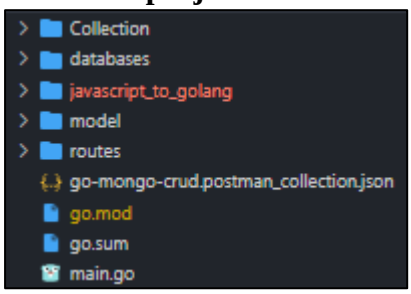
Task3-TEFA
CRUD 2 features with Golang

Nama anggota kelompok:

Helsa Nesta Dhaifullah - 5025201005
Naily Khairiya - 5025201244

We chose Helsa's previous task, it is an e-learning app, and we create CRUD course and CRUD material with Golang language, Gin framework, and MongoDB as the database. We also try to implement hash table in our source code.
You can see the complete source code in (link github)

- **Structure project file**



Collection is to handle the collection-fetching functionality.
Databases is config for connect with our database in MongoDB.
Model is for the data struct of course and material for database.
Routes is for the controller of CRUD.
Main.go is the main program.

- **Connect Go to MongoDB**

```go
func ConnectDB() *mongo.Client {
    Mongo_URL := "mongodb://localhost:27017"
    client, err := mongo.NewClient(options.Client().ApplyURI(Mongo_URL))

    if err != nil {
        log.Fatal(err)
    }

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    err = client.Connect(ctx)
```

The ConnectDB function establishes a connection and returns a new MongoDB Client object. But first, you need setup the mongoDB, and copy-paste the MongoDB URl to connect Golang with the database.

- **Create Database collection**

```go
package getcollection

import (
    "go.mongodb.org/mongo-driver/mong
)   o"

func GetCollection(client *mongo.Client, collectionName string) *mongo.Collection {
    collection := client.Database("myGoappDB").Collection(collectionName)
    return collection
}
```

This function gets the Collection from the MongoDB database. The database name, in this case, is myGoappDB, with collectionName as its collection.

- **Database model**

```
type Course struct {
    ID          primitive.ObjectI
D   CourseName  string
    Description string
    CreatedAt   time.Time
    UpdatedAt   time.Time
}
```

We have 5 components in Course, such as ID(primitive object), CourseName (string), Description (string), CreatedAt(time), and UpdatedAt(time).

```
type Material struct {
    ID           primitive.ObjectID
    CourseName   string
    MaterialName string
    Description  string
    CreatedAt    time.Time
    UpdatedAt    time.Time
}
```

In our model, we create a struct for the material model as well. We have 6 components in material: ID(primitive object), CourseName (string), MaterialName(string), Description (string), CreatedAt(time), and UpdatedAt(time).

A. CRUD Course

- POST create course

```
// Use map to store course payloa
d   data := make(map[string]interface{})
    data["id"] = coursePayload.ID.Hex()
    data["coursename"] = coursePayload.CourseName
    data["description"] = coursePayload.Description
    data["created_at"] = coursePayload.CreatedAt
    data["updated_at"] = coursePayload.UpdatedAt

    // Store course payload in hash tabl
    esult, err := courseCollection.InsertOne(ctx, data)
```

The function of the CreateCourse method is to create a new data course in the MongoDB database using the Gin framework. The course data to be created will be taken from the JSON request sent by the client. Then, the data will be stored in a folder and will be included in a collection in the MongoDB database. By using a map, the data can be easily accessed and manipulated by key rather than having to access each field in the coursePayload struct.

- GET all courses

```
var courses []map[string]interface{}
    for cursor.Next(ctx) {
        var course model.Course
        err := cursor.Decode(&course)
        if err != nil {
            c.JSON(http.StatusInternalServerError, gin.H "message": err})
            return{
        }

        // Store course data in a map
        data := make(map[string]interface{})
        data["id"] = course.ID.Hex()
        data["coursename"] = course.CourseName
        data["description"] = course.Description
        data["created_at"] = course.CreatedAt
        data["updated_at"] = course.UpdatedAt

        courses = append(courses, data)
    }
```

This function of the GetAllCourses method is to handle a GET request to retrieve all courses. It then iterates through the cursor and stores the course data in a map using the make function to create a new map, and then appends the data to a slice of maps called courses. The slice of maps can be returned as a JSON response to the client.

- GET one course

```
courseID := c.Param("courseID")
    var result model.Course

    objId, _ := primitive.ObjectIDFromHex(courseID)

    err := courseCollection.FindOne(ctx, bson.M{"_id": objId}).Decode(&result)

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H "message": err})
        return{
    }

    res := map[string]interface{}{"data": result}
```

This code defines a function that retrieves a single course from a MongoDB database based on its ID. The courseID parameter is retrieved from the URL path using the Param() method of the gin.Context object. A variable of type model.Course is created to store the result of the database query, and the ID string is converted to a MongoDB ObjectID. The FindOne() method is then called on the courseCollection object to retrieve the course with the specified ID, and the result is stored in the previously created variable.

- PUT update course

```
objId, _ := primitive.ObjectIDFromHex(courseID)

    if err := c.BindJSON(&course); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H "message": err})
        return{
    }

    edited := bson.M{"coursename": course.CourseName, "description": course.Description}
    edited["updatedAt"] = time.Now()

    result, err := courseCollection.UpdateOne(ctx, bson.M{"_id": objId}, bson.M{"$set"
: edited})

    res := map[string]interface{}{"data": result}
```

The code is an endpoint function to update a course in a MongoDB collection. The code then gets the courseID parameter from the request and decodes it into an ObjectId. The edited map is created with the fields that can be edited in the course document, with the updatedAt field being set to the current time. The courseCollection.UpdateOne method is then called, passing in the filter and update documents created from the ObjectId and edited map. The result of the update is returned and stored in the res map.

- DELETE course

```
courseID := c.Param("courseID")

    objID, err := primitive.ObjectIDFromHex(courseID)
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H "message": "Invalid ID"})
        return{
    }

    result, err := courseCollection.DeleteOne(ctx, bson.M{"_id": objID})
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H "message": err})
        return{
    }
```

The code defines a function named DeleteCourse that receives a courseID as a parameter from the URL path and converts it into an object ID. The function then deletes the document with the given ID from the MongoDB collection.

- Routes in main.go

```
router.POST("/courses", routes.CreateCourse)

// called as localhost:3000/getAl
router.GET("/courses", routes.GetAllCourses)

// called as localhost:3000/get/{i
router.GET("/course/:courseID", routes.ReadOneCourse)

// called as localhost:3000/update/{i
router.PUT("/course/:courseID", routes.UpdateCourse)

// called as localhost:3000/delete/{i
router.DELETE("/course/:courseID", routes.DeleteCourse)

router.Run("localhost: 3000")
```

The router is defined to handle various HTTP requests with different routes to different handler functions.

- POST request with the route "/courses" is mapped to the **CreateCourse** handler function for create new course.
- GET request with route "/courses" is mapped to **GetAllCourses** handler function to retrieve and display all courses.
- GET request with route "/course/:courseID" is mapped to **ReadOneCourse** handler function to retrieve a single course based on course ID.
- PUT request with route "/course/:courseID" is mapped to **UpdateCourse** handler function to update an existing course based on course ID.
  DELETE request with route "/course/:courseID" is mapped to **DeleteCourse** handler function to delete an existing course based on course ID

B. CRUD Material

❖ **CREATE NEW MATERIAL**

```go
func CreateMaterial(c *gin.Context) {
    var DB = database.ConnectDB()
    var materialCollection = getcollection.GetCollection(DB, "Materials")
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    material := new(model.Material)
    defer cancel()

    if err := c.BindJSON(&material); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"message": err})
        log.Fatal(err)
        return
    }

    materialPayload := model.Material{
        ID:           primitive.NewObjectID(),
        CourseName:   material.CourseName,
        MaterialName: material.MaterialName,
        Description:  material.Description,
        CreatedAt:    time.Now(),
        UpdatedAt:    time.Now(),
    }

    // use map  to store material payload
    data := make(map[string]interface{})
    data["id"] = materialPayload.ID.Hex()
    data["coursename"] = materialPayload.CourseName
    data["materialname"] = material.MaterialName
    data["description"] = materialPayload.Description
    data["created_at"] = materialPayload.CreatedAt
    data["updated_at"] = materialPayload.UpdatedAt

    result, err := materialCollection.InsertOne(ctx, materialPayload)

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    c.JSON(http.StatusCreated, gin.H{"message": "Material upload successfully", "Data": map[string]interface{}{"data":
result}})
}
```

The function of the CreateMaterial method is to create a new data material in the MongoDB database using the Gin framework. The material data to be created will be taken from the JSON request sent by the client. Then, the data will be stored in a folder and will be included in a collection in the MongoDB database. By using a map, the data can be easily accessed and manipulated by key rather than having to access each field in the materialPayload struct.

- **GET ALL MATERIAL**

```go
func GetAllMaterial(c *gin.Context) {
    ctx, cancel := context.WithTimeout(context.Background(),
10*time.Second)
    defer cancel()

    var DB = database.ConnectDB()
    var materialCollection = getcollection.GetCollection(DB, "Materials")

    cursor, err := materialCollection.Find(ctx, bson.M{})
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }
    defer cursor.Close(ctx)

    var materials []map[string]interface{}
    for cursor.Next(ctx) {
        var material model.Material
        err := cursor.Decode(&material)
        if err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"message": err})
            return
        }

        // Store course data in a map
        data := make(map[string]interface{})
        data["id"] = material.ID.Hex()
        data["coursename"] = material.CourseName
        data["materialname"] = material.MaterialName
        data["description"] = material.Description
        data["created_at"] = material.CreatedAt
        data["updated_at"] = material.UpdatedAt

        materials = append(materials, data)
    }

    if err := cursor.Err(); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "success", "Data": materials})
}
```

This function of the GetAllMaterial method is to handle a GET request to retrieve all materials. It then iterates through the cursor and stores the material data in a map using the make function to create a new map, and then appends the data to a slice of maps called materials. The slice of maps can be returned as a JSON response to the client.

- **GET A MATERIAL BY ID**

```go
func GetOneMaterial(c *gin.Context) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    var DB = database.ConnectDB()
    var materialCollection = getcollection.GetCollection(DB, "Materials")

    materialID := c.Param("materialID")
    var result model.Material

    objId, _ := primitive.ObjectIDFromHex(materialID)

    err := materialCollection.FindOne(ctx, bson.M{"id":
objId}).Decode(&result)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    res := map[string]interface{}{"data": result}
    c.JSON(http.StatusOK, gin.H{"message": "success!", "Data": res})
}
```

This code defines a function that retrieves a single material from a MongoDB database based on its ID. The materialID parameter is retrieved from the URL path using the Param() method of the gin.Context object. A variable of type model.Material is created to store the result of the database query, and the ID string is converted to a MongoDB ObjectID. The GetOneMaterial() method is then called on the materialCollection object to retrieve the material with the specified ID, and the result is stored in the previously created variable.

❖ UPDATE A MATERIAL BY ID

```go
func UpdateMaterial(c *gin.Context) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    var DB = database.ConnectDB()
    var materialCollection = getcollection.GetCollection(DB, "Materials")

    materialID := c.Param("materialID")
    var material model.Material

    defer cancel()

    objId, _ := primitive.ObjectIDFromHex(materialID)

    if err := c.BindJSON(&material); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    edited := bson.M{"CourseName": material.CourseName, "MaterialName": material.MaterialName, "description":
material.Description}
    edited["updatedAt"] = time.Now()

    result, err := materialCollection.UpdateOne(ctx, bson.M{"id": objId}, bson.M{"$set": edited})

    res := map[string]interface{}{"data": result}

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    if result.MatchedCount < 1 {
        c.JSON(http.StatusInternalServerError, gin.H{"message": "Data doesn't exist"})
        return
    }

    c.JSON(http.StatusCreated, gin.H{"message": "data updated successfully!", "Data": res})
}
```

The code is an endpoint function to update a material in a MongoDB collection. The code then gets the materialID parameter from the request and decodes it into an ObjectId. The edited map is created with the fields that can be edited in the material document, with the updatedAt field being set to the current time. The materialCollection.UpdateMaterial method is then called, passing in the filter and update documents created from the ObjectId and edited map. The result of the update is returned and stored in the res map.

❖ **DELETE A MATERIAL BY ID**

```go
func DeleteMaterial(c *gin.Context) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    var DB = database.ConnectDB()
    materialID := c.Param("materialID")

    var materialCollection = getcollection.GetCollection(DB, "Materials")
    defer cancel()

    objId, _ := primitive.ObjectIDFromHex(materialID)
    result, err := materialCollection.DeleteOne(ctx, bson.M{"id": objId})
    res := map[string]interface{}{"data": result}

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"message": err})
        return
    }

    if result.DeletedCount < 1 {
        c.JSON(http.StatusInternalServerError, gin.H{"message": "No data to delete"})
        return
    }

    c.JSON(http.StatusCreated, gin.H{"message": "Material deleted successfully", "Data": res})
}
```

The code defines a function named DeleteMaterial that receives a materialID as a parameter from the URL path and converts it into an object ID. The function then deletes the document with the given ID from the MongoDB collection.

❖ **END POINT**

```go
    /* MATERIALS */
    // called as localhost:3000/materials
    router.GET("/materials", routes.GetAllMaterial)

    // called as localhost:3000/materials/:id
    router.GET("/material/:materialID", routes.GetOneMaterial)

    // called as localhost:3000/materials
    router.POST("/materials", routes.CreateMaterial)

    // called as localhost:3000/materials/:id
    router.PUT("/material/:materialID", routes.UpdateMaterial)

    // called as localhost:3000/materials/:id
    router.DELETE("/material/:materialID",
routes.DeleteMaterial)
```

The router is defined to handle various HTTP requests with different routes to different handler functions.

- **POST** request with the route "/materials" is mapped to the **CreateMaterial** handler function for create new material.
- **GET** request with route "/materials" is mapped to **GetAllMaterial** handler function to retrieve and display all materials.
- **GET** request with route "/material/:materialID" is mapped to **GetOneMaterial** handler function to retrieve a single material based on course ID.
- **PUT** request with route "/material/:materialID" is mapped to **UpdateMaterial** handler function to update an existing material based on material ID.
- **DELETE** request with route "/material/:materialID" is mapped to **DeleteMaterial** handler function to delete an existing material based on material ID.

C. BENCHMARK CRUD COURSE
#HASHTABLE LOAD TESTING USING HEY PACKAGE

- **Create course**

10 request

```
$ hey -m POST -n 10 -c 10 -d '{"CourseName": "Test Course", "Description": "This is a test course."}' http://localhost:3000/

Summary:
  Total:        0.9569 secs
  Slowest:      0.9566 secs
  Fastest:      0.9429 secs
  Average:      0.9506 secs
  Requests/sec: 10.4502

  Total data:   910 bytes
  Size/request: 91 bytes

Response time histogram:
  0.943 [1]     |■■■■■■■■■■■■■
  0.944 [1]     |■■■■■■■■■■■■■
  0.946 [1]     |■■■■■■■■■■■■■
  0.947 [0]     |
  0.948 [0]     |
  0.950 [1]     |■■■■■■■■■■■■■
  0.951 [1]     |■■■■■■■■■■■■■
  0.952 [0]     |
  0.954 [0]     |
  0.955 [3]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.957 [2]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9429 secs
  25% in 0.9496 secs
  50% in 0.9542 secs
  75% in 0.9554 secs
  90% in 0.9566 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3184 secs, 0.9429 secs, 0.9566 secs
  DNS-lookup:   0.0048 secs, 0.0041 secs, 0.0055 secs
  req write:    0.0003 secs, 0.0000 secs, 0.0009 secs
  resp wait:    0.6317 secs, 0.6245 secs, 0.6392 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0002 secs

Status code distribution:
  [201] 10 responses
```

100 request

```
$ hey -m POST -n 100 -c 10 -d '{"CourseName": "Test Course", "Description": "This is a test course."}' http://localhost:3000/courses

Summary:
  Total:        6.6931 secs
  Slowest:      0.9558 secs
  Fastest:      0.6151 secs
  Average:      0.6681 secs
  Requests/sec: 14.9408

  Total data:   9100 bytes
  Size/request: 91 bytes

Response time histogram:
  0.615 [1]     |
  0.649 [83]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.683 [6]     |■■■
  0.717 [0]     |
  0.751 [0]     |
  0.785 [0]     |
  0.820 [0]     |
  0.854 [0]     |
  0.888 [0]     |
  0.922 [0]     |
  0.956 [10]    |■■■■■


Latency distribution:
  10% in 0.6237 secs
  25% in 0.6314 secs
  50% in 0.6391 secs
  75% in 0.6447 secs
  90% in 0.9476 secs
  95% in 0.9533 secs
  99% in 0.9558 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0311 secs, 0.6151 secs, 0.9558 secs
  DNS-lookup:   0.0008 secs, 0.0000 secs, 0.0082 secs
  req write:    0.0002 secs, 0.0000 secs, 0.0114 secs
  resp wait:    0.6367 secs, 0.6150 secs, 0.6527 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0004 secs

Status code distribution:
  [201] 100 responses
```

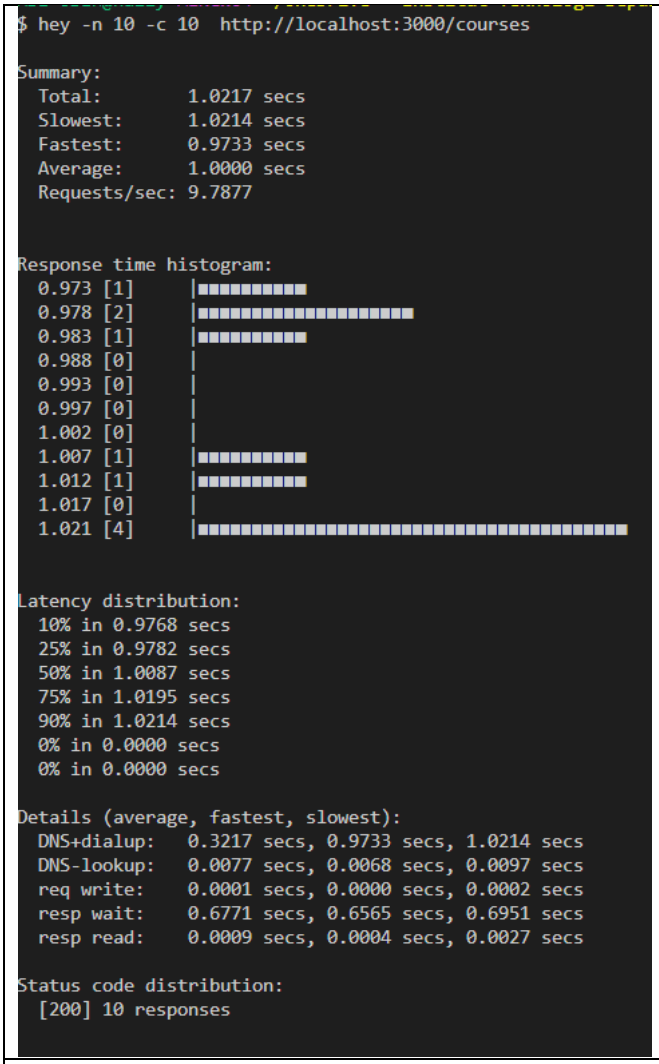
1000 request

```
MSI USER@naily MINGW64 ~/OneDrive - Institut Teknologi Sepuluh Nopember/Dokumen/TELKOM/TUGAS/Task3-TEFA-golang-crud
$ hey -m POST -n 1000 -c 100 -d '{"CourseName": "Test Course", "Description": "This is a test course."}' http://loca

Summary:
  Total:        10.9113 secs
  Slowest:      1.4048 secs
  Fastest:      0.6359 secs
  Average:      1.0761 secs
  Requests/sec: 91.6482

  Total data:   91000 bytes
  Size/request: 91 bytes

Response time histogram:
  0.636 [1]     |
  0.713 [8]     |■
  0.790 [6]     |■
  0.867 [13]    |■■
  0.943 [100]   |■■■■■■■■■■■■■■■
  1.020 [197]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.097 [261]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.174 [211]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.251 [114]   |■■■■■■■■■■■■■■■■■■
  1.328 [60]    |■■■■■■■■■
  1.405 [29]    |■■■■


Latency distribution:
  10% in 0.9281 secs
  25% in 0.9876 secs
  50% in 1.0711 secs
  75% in 1.1505 secs
  90% in 1.2331 secs
  95% in 1.3126 secs
  99% in 1.3888 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0355 secs, 0.6359 secs, 1.4048 secs
  DNS-lookup:   0.0011 secs, 0.0000 secs, 0.0148 secs
  req write:    0.0008 secs, 0.0000 secs, 0.0604 secs
  resp wait:    1.0390 secs, 0.6357 secs, 1.2849 secs
  resp read:    0.0007 secs, 0.0000 secs, 0.0505 secs

Status code distribution:
  [201] 1000 responses
```

Conclusion :
Average time for get all course 1s.

- **Get all courses**

10 request


$ hey -n 10 -c 10 http://localhost:3000/courses
Summary:
Total: 1.0217 secs
Slowest: 1.0214 secs
Fastest: 0.9733 secs
Average: 1.0000 secs
Requests/sec: 9.7877
Response time histogram:
0.973 [1]
0.978 [2]
0.983 [1]
0.988 [0]
0.993 [0]
0.997 [0]
1.002 [0]
1.007 [1]
1.012 [1]
1.017 [0]
1.021 [4]
Latency distribution:
10% in 0.9768 secs
25% in 0.9782 secs
50% in 1.0087 secs
75% in 1.0195 secs
90% in 1.0214 secs
0% in 0.0000 secs
0% in 0.0000 secs
Details (average, fastest, slowest):
DNS+dialup: 0.3217 secs, 0.9733 secs, 1.0214 secs
DNS-lookup: 0.0077 secs, 0.0068 secs, 0.0097 secs
req write: 0.0001 secs, 0.0000 secs, 0.0002 secs
resp wait: 0.6771 secs, 0.6565 secs, 0.6951 secs
resp read: 0.0009 secs, 0.0004 secs, 0.0027 secs
Status code distribution:
[200] 10 responses

100 request

```
$ hey -n 100 -c 100  http://localhost:3000/courses

Summary:
  Total:        2.3061 secs
  Slowest:      2.3052 secs
  Fastest:      1.5647 secs
  Average:      2.2181 secs
  Requests/sec: 43.3635


Response time histogram:
  1.565 [1]     |
  1.639 [2]     |■
  1.713 [0]     |
  1.787 [3]     |■
  1.861 [3]     |■
  1.935 [3]     |■
  2.009 [0]     |
  2.083 [1]     |
  2.157 [2]     |■
  2.231 [3]     |■
  2.305 [82]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 1.8759 secs
  25% in 2.2486 secs
  50% in 2.2970 secs
  75% in 2.3009 secs
  90% in 2.3036 secs
  95% in 2.3043 secs
  99% in 2.3052 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.4350 secs, 1.5647 secs, 2.3052 secs
  DNS-lookup:   0.0579 secs, 0.0070 secs, 0.1532 secs
  req write:    0.0041 secs, 0.0000 secs, 0.0424 secs
  resp wait:    1.7254 secs, 1.2548 secs, 1.9003 secs
  resp read:    0.0535 secs, 0.0001 secs, 0.1194 secs

Status code distribution:
  [200] 100 responses
```

1000 request

```
$ hey -n 1000 -c 100  http://localhost:3000/courses

Summary:
  Total:        14.7515 secs
  Slowest:      2.2615 secs
  Fastest:      0.7251 secs
  Average:      1.3907 secs
  Requests/sec: 67.7895


Response time histogram:
  0.725 [1]     |
  0.879 [10]    |■■
  1.032 [68]    |■■■■■■■■■■■■■
  1.186 [200]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.340 [206]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.493 [215]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.647 [107]   |■■■■■■■■■■■■■■■■■■■■
  1.801 [56]    |■■■■■■■■■■
  1.954 [89]    |■■■■■■■■■■■■■■■■■
  2.108 [43]    |■■■■■■■■
  2.261 [5]     |■


Latency distribution:
  10% in 1.0556 secs
  25% in 1.1663 secs
  50% in 1.3511 secs
  75% in 1.5448 secs
  90% in 1.8879 secs
  95% in 1.9483 secs
  99% in 2.0834 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0399 secs, 0.7251 secs, 2.2615 secs
  DNS-lookup:   0.0031 secs, 0.0000 secs, 0.0639 secs
  req write:    0.0005 secs, 0.0000 secs, 0.0702 secs
  resp wait:    1.3155 secs, 0.7244 secs, 2.2430 secs
  resp read:    0.0347 secs, 0.0001 secs, 0.3098 secs

Status code distribution:
  [200] 1000 responses
```

Conclusion :
Average performance is 1s

- **Get a course**

10 request

```
$ hey -n 10 -c 10 http://localhost:3000/course/6436b7e17b2360fedb6e2181

Summary:
  Total:        1.1358 secs
  Slowest:      1.1350 secs
  Fastest:      1.0189 secs
  Average:      1.1080 secs
  Requests/sec: 8.8047

  Total data:   1850 bytes
  Size/request: 185 bytes

Response time histogram:
  1.019 [1]     |■■■■■■■
  1.031 [0]     |
  1.042 [0]     |
  1.054 [0]     |
  1.065 [0]     |
  1.077 [0]     |
  1.089 [2]     |■■■■■■■■■■■■■
  1.100 [1]     |■■■■■■■
  1.112 [0]     |
  1.123 [0]     |
  1.135 [6]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 1.0775 secs
  25% in 1.0976 secs
  50% in 1.1342 secs
  75% in 1.1349 secs
  90% in 1.1350 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3426 secs, 1.0189 secs, 1.1350 secs
  DNS-lookup:   0.0122 secs, 0.0109 secs, 0.0179 secs
  req write:    0.0001 secs, 0.0001 secs, 0.0004 secs
  resp wait:    0.7650 secs, 0.7032 secs, 0.8138 secs
  resp read:    0.0002 secs, 0.0000 secs, 0.0007 secs

Status code distribution:
  [200] 10 responses
```

100 request

```
$ hey -n 100 -c 10 http://localhost:3000/course/6436b7e17b2360fedb6e2181

Summary:
  Total:        7.1857 secs
  Slowest:      1.2144 secs
  Fastest:      0.6148 secs
  Average:      0.7063 secs
  Requests/sec: 13.9166

  Total data:   18500 bytes
  Size/request: 185 bytes

Response time histogram:
  0.615 [1]     |■
  0.675 [70]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.735 [12]    |■■■■■■■
  0.795 [7]     |■■■■
  0.855 [0]     |
  0.915 [0]     |
  0.975 [0]     |
  1.035 [0]     |
  1.095 [0]     |
  1.154 [2]     |■
  1.214 [8]     |■■■■■


Latency distribution:
  10% in 0.6220 secs
  25% in 0.6299 secs
  50% in 0.6404 secs
```

```
  75% in 0.6873 secs
  90% in 1.1300 secs
  95% in 1.1859 secs
  99% in 1.2144 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0338 secs, 0.6148 secs, 1.2144 secs
  DNS-lookup:   0.0013 secs, 0.0000 secs, 0.0148 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0003 secs
  resp wait:    0.6722 secs, 0.6145 secs, 0.8714 secs
  resp read:    0.0002 secs, 0.0000 secs, 0.0022 secs

Status code distribution:
  [200] 100 responses
```

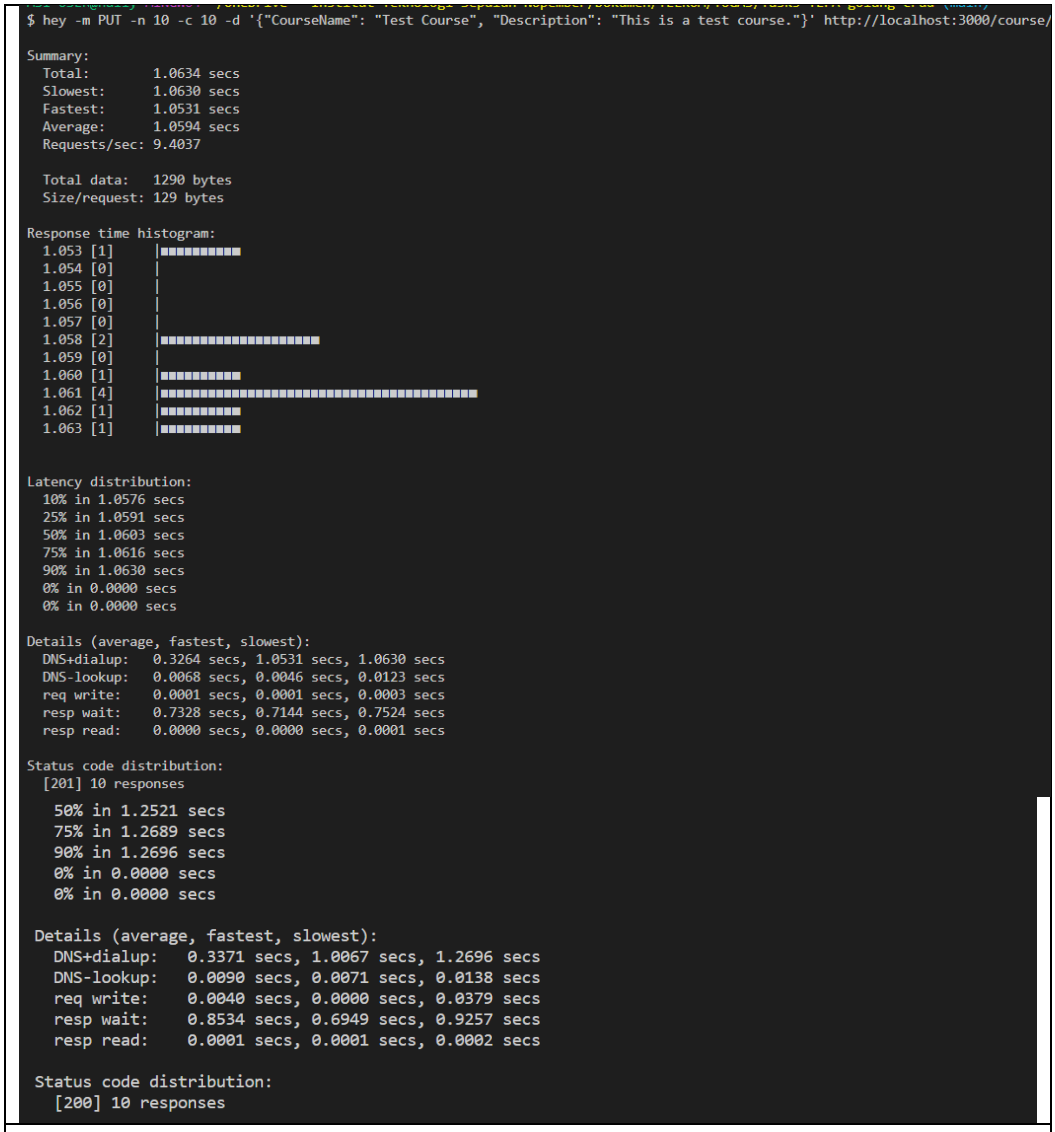
1000 request:

```
$ hey -n 1000 -c 100 http://localhost:3000/course/6436b7e17b2360fedb6e2181

Summary:
  Total:        16.6390 secs
  Slowest:      3.5672 secs
  Fastest:      0.9005 secs
  Average:      1.6448 secs
  Requests/sec: 60.0997

  Total data:   185000 bytes
  Size/request: 185 bytes

Response time histogram:
  0.901 [1]     |
  1.167 [39]    |■■■
  1.434 [216]   |■■■■■■■■■■■■■■■
  1.701 [591]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.967 [51]    |■■■
  2.234 [2]     |
  2.501 [0]     |
  2.767 [5]     |
  3.034 [71]    |■■■■■
  3.301 [17]    |■
  3.567 [7]     |


Latency distribution:
  10% in 1.3247 secs
  25% in 1.4309 secs
  50% in 1.5262 secs
  75% in 1.6255 secs
  90% in 2.5599 secs
  95% in 2.9731 secs
  99% in 3.1099 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0459 secs, 0.9005 secs, 3.5672 secs
  DNS-lookup:   0.0047 secs, 0.0000 secs, 0.1258 secs
  req write:    0.0003 secs, 0.0000 secs, 0.0568 secs
  resp wait:    1.5985 secs, 0.9004 secs, 3.0398 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0017 secs

Status code distribution:
  [200] 1000 responses
```

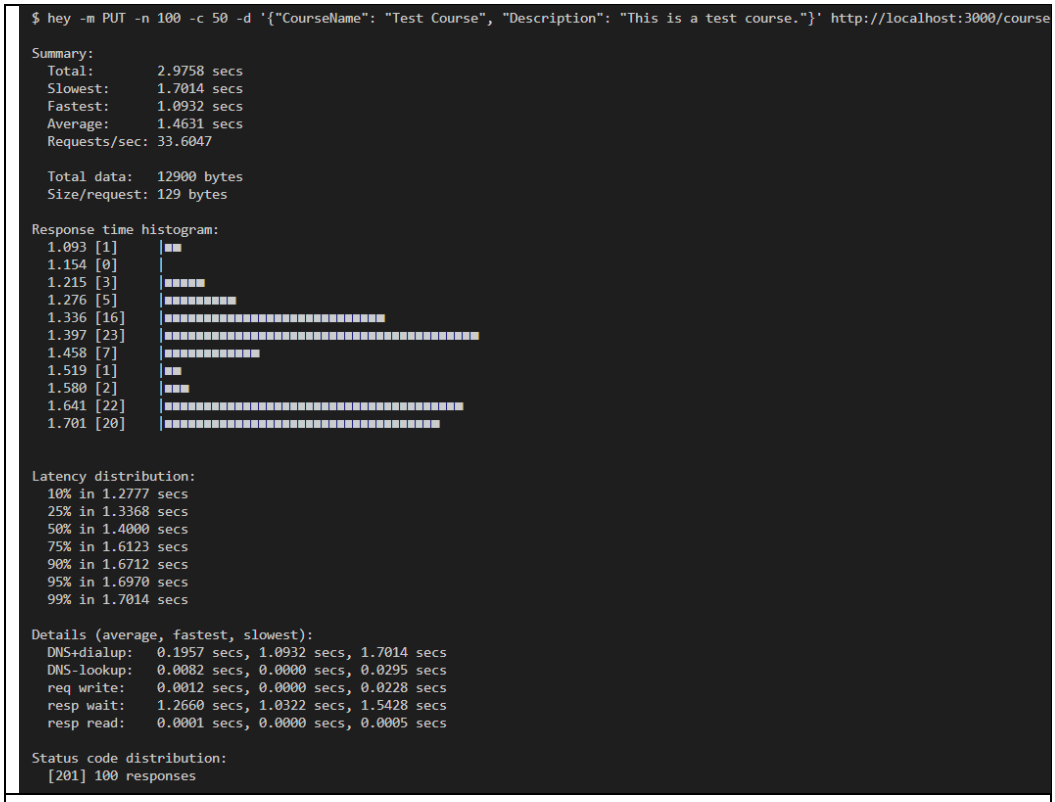
Conclusion:
For 10, 100, 1000 request each of request has average around 1s.

- **Update course**

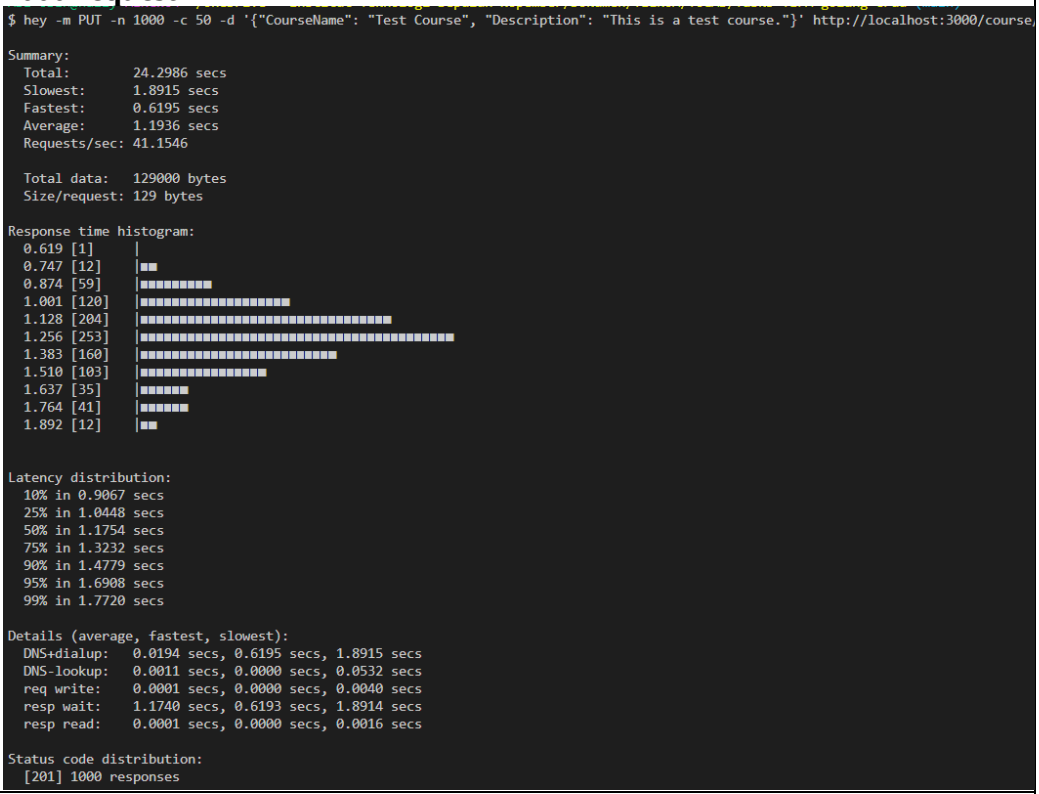
10 request


$ hey -m PUT -n 10 -c 10 -d '{"CourseName": "Test Course", "Description": "This is a test course."}' http://localhost:3000/course/
Summary:
Total: 1.0634 secs
Slowest: 1.0630 secs
Fastest: 1.0531 secs
Average: 1.0594 secs
Requests/sec: 9.4037
Total data: 1290 bytes
Size/request: 129 bytes
Response time histogram:
1.053 [1]
1.054 [0]
1.055 [0]
1.056 [0]
1.057 [0]
1.058 [2]
1.059 [0]
1.060 [1]
1.061 [4]
1.062 [1]
1.063 [1]
Latency distribution:
10% in 1.0576 secs
25% in 1.0591 secs
50% in 1.0603 secs
75% in 1.0616 secs
90% in 1.0630 secs
0% in 0.0000 secs
0% in 0.0000 secs
Details (average, fastest, slowest):
DNS+dialup: 0.3264 secs, 1.0531 secs, 1.0630 secs
DNS-lookup: 0.0068 secs, 0.0046 secs, 0.0123 secs
req write: 0.0001 secs, 0.0001 secs, 0.0003 secs
resp wait: 0.7328 secs, 0.7144 secs, 0.7524 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0001 secs
Status code distribution:
[201] 10 responses
50% in 1.2521 secs
75% in 1.2689 secs
90% in 1.2696 secs
0% in 0.0000 secs
0% in 0.0000 secs
Details (average, fastest, slowest):
DNS+dialup: 0.3371 secs, 1.0067 secs, 1.2696 secs
DNS-lookup: 0.0090 secs, 0.0071 secs, 0.0138 secs
req write: 0.0040 secs, 0.0000 secs, 0.0379 secs
resp wait: 0.8534 secs, 0.6949 secs, 0.9257 secs
resp read: 0.0001 secs, 0.0001 secs, 0.0002 secs
Status code distribution:
[200] 10 responses

100 request



1000 request



Conclusion:

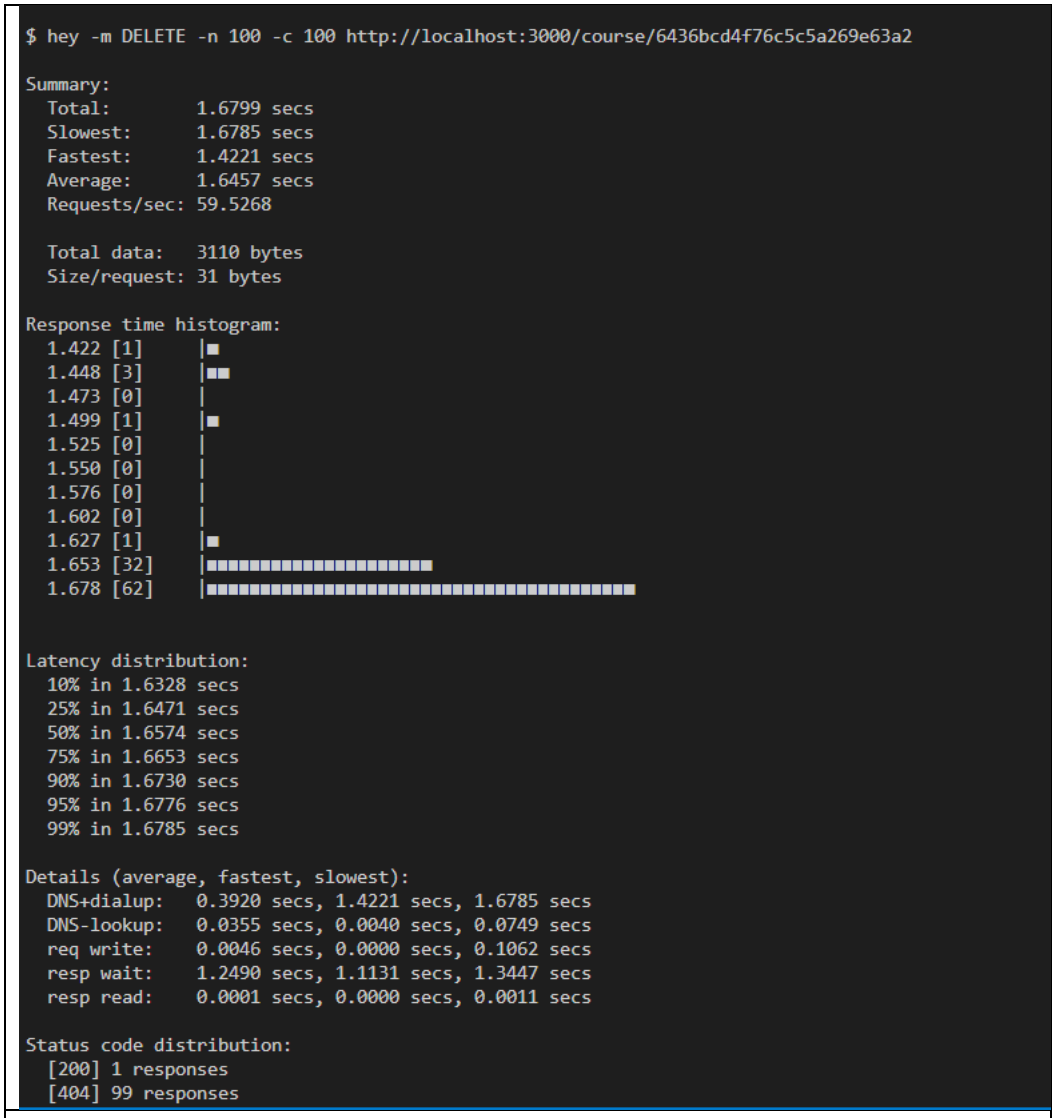
For 10, 100, 1000 request each of request has average around 1s.

- **Delete course**

10 request

```
$ hey -m DELETE -n 10 -c 10 http://localhost:3000/course/6436bcfaf76c5c5a269e6409

Summary:
  Total:        0.9811 secs
  Slowest:      0.9807 secs
  Fastest:      0.9562 secs
  Average:      0.9766 secs
  Requests/sec: 10.1925

  Total data:   320 bytes
  Size/request: 32 bytes

Response time histogram:
  0.956 [1]     |■■■■■■
  0.959 [0]     |
  0.961 [0]     |
  0.964 [0]     |
  0.966 [0]     |
  0.968 [0]     |
  0.971 [0]     |
  0.973 [1]     |■■■■■■
  0.976 [0]     |
  0.978 [1]     |■■■■■■
  0.981 [7]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9721 secs
  25% in 0.9789 secs
  50% in 0.9796 secs
  75% in 0.9804 secs
  90% in 0.9807 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3170 secs, 0.9562 secs, 0.9807 secs
  DNS-lookup:   0.0056 secs, 0.0051 secs, 0.0064 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0004 secs
  resp wait:    0.6593 secs, 0.6426 secs, 0.6717 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0002 secs

Status code distribution:
  [200] 1 responses
  [404] 9 responses
```

100 request



1000 request

```
$ hey -m DELETE -n 1000 -c 100 http://localhost:3000/course/6436b7e17b2360fedb6e2181

Summary:
  Total:        10.7725 secs
  Slowest:      1.5127 secs
  Fastest:      0.6163 secs
  Average:      1.0384 secs
  Requests/sec: 92.8290

  Total data:   31010 bytes
  Size/request: 31 bytes

Response time histogram:
  0.616 [1]     |
  0.706 [17]    |■■■
  0.796 [81]    |■■■■■■■■■■■■■
  0.885 [115]   |■■■■■■■■■■■■■■■■■■■
  0.975 [114]   |■■■■■■■■■■■■■■■■■■■
  1.064 [220]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.154 [246]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.244 [101]   |■■■■■■■■■■■■■■■■
  1.333 [23]    |■■■■
  1.423 [27]    |■■■■
  1.513 [55]    |■■■■■■■■■


Latency distribution:
  10% in 0.7961 secs
  25% in 0.9085 secs
  50% in 1.0374 secs
  75% in 1.1357 secs
  90% in 1.2558 secs
  95% in 1.4314 secs
  99% in 1.5037 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0384 secs, 0.6163 secs, 1.5127 secs
  DNS-lookup:   0.0028 secs, 0.0000 secs, 0.0523 secs
  req write:    0.0004 secs, 0.0000 secs, 0.0550 secs
  resp wait:    0.9994 secs, 0.6159 secs, 1.3731 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0022 secs

Status code distribution:
  [200] 1 responses
  [404] 999 responses
```

Conclusion:
For 10, 100, 1000 request each of request has average around 1s.

D. BENCHMARK CRUD MATERIAL
HASHTABLE LOAD TESTING USING HEY PACKAGE

- **CREATE MATERIAL**

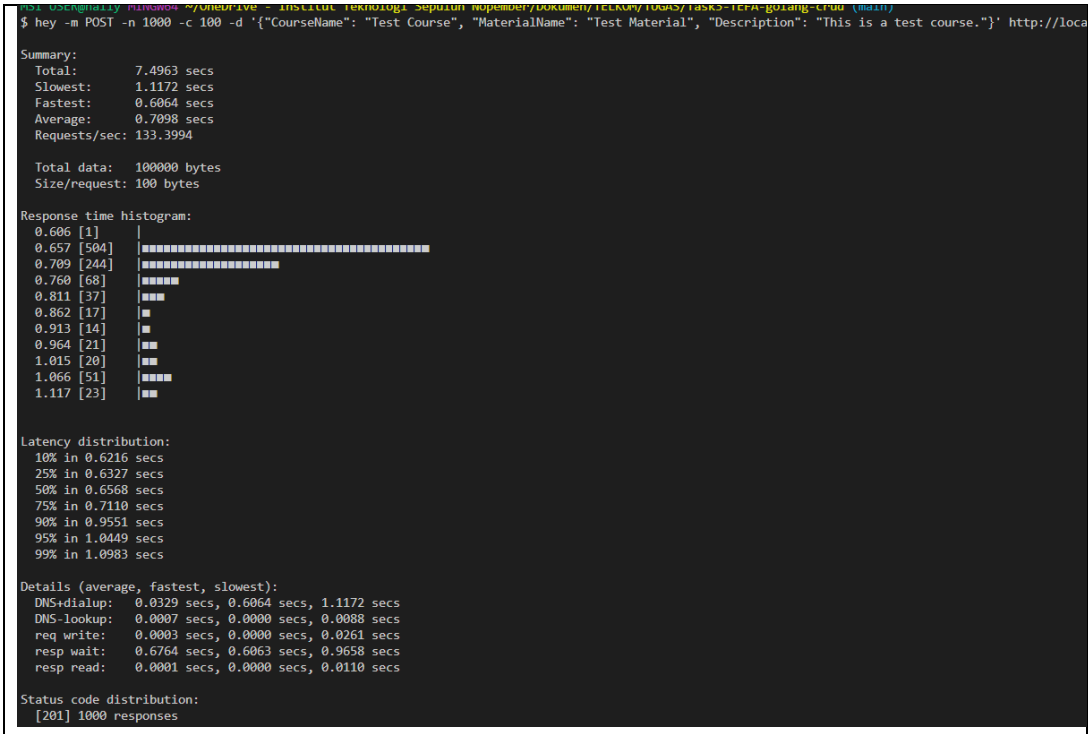
10 request

```
MSI USER@Hally MINGW64 ~/OneDrive - Institut Teknologi Sepuluh Nopember/Dokumen/TELKOM/TUGAS/Task3-TEFA-golang-crud (main)
$ hey -m POST -n 10 -c 10 -d '{"CourseName": "Test Course", "MaterialName": "Test Material", "Description": "This is a test course."}' http://lo

Summary:
  Total:        1.0293 secs
  Slowest:      1.0290 secs
  Fastest:      1.0100 secs
  Average:      1.0253 secs
  Requests/sec: 9.7155

  Total data:   1000 bytes
  Size/request: 100 bytes

Response time histogram:
  1.010 [1]     |■■■■■■
  1.012 [0]     |
  1.014 [0]     |
  1.016 [0]     |
  1.018 [1]     |■■■■■■
  1.020 [0]     |
  1.021 [0]     |
  1.023 [0]     |
  1.025 [0]     |
  1.027 [1]     |■■■■■■
  1.029 [7]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 1.0163 secs
  25% in 1.0282 secs
  50% in 1.0288 secs
  75% in 1.0289 secs
  90% in 1.0290 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3225 secs, 1.0100 secs, 1.0290 secs
  DNS-lookup:   0.0043 secs, 0.0041 secs, 0.0045 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0003 secs
  resp wait:    0.7026 secs, 0.6918 secs, 0.7131 secs
  resp read:    0.0000 secs, 0.0000 secs, 0.0001 secs

Status code distribution:
  [201] 10 responses
```

100 request

```
$ hey -m POST -n 100 -c 100 -d '{"CourseName": "Test Course", "MaterialName": "Test Material", "Description": "This is a test course."}' http://loc

Summary:
  Total:        1.0639 secs
  Slowest:      1.0628 secs
  Fastest:      0.9376 secs
  Average:      1.0214 secs
  Requests/sec: 93.9901

  Total data:   10000 bytes
  Size/request: 100 bytes

Response time histogram:
  0.938 [1]     |■
  0.950 [9]     |■■■■■■■■■■■■■
  0.963 [0]     |
  0.975 [9]     |■■■■■■■■■■■■■
  0.988 [10]    |■■■■■■■■■■■■■■
  1.000 [0]     |
  1.013 [0]     |
  1.025 [6]     |■■■■■■■■■
  1.038 [14]    |■■■■■■■■■■■■■■■■■■■■
  1.050 [28]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.063 [23]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9705 secs
  25% in 0.9813 secs
  50% in 1.0421 secs
  75% in 1.0497 secs
  90% in 1.0609 secs
  95% in 1.0622 secs
  99% in 1.0628 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3184 secs, 0.9376 secs, 1.0628 secs
  DNS-lookup:   0.0065 secs, 0.0038 secs, 0.0081 secs
  req write:    0.0028 secs, 0.0000 secs, 0.0101 secs
  resp wait:    0.7000 secs, 0.6224 secs, 0.7399 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0005 secs

Status code distribution:
  [201] 100 responses
```

1000 request



Conclusion:
For 10, 100, 1000 request each of request has average around 1s.

- **GET ALL MATERIAL**

10 requests

```
$ hey -n 10 -c 10 http://localhost:3000/materials

Summary:
  Total:        0.9401 secs
  Slowest:      0.9400 secs
  Fastest:      0.9386 secs
  Average:      0.9392 secs
  Requests/sec: 10.6368

  Total data:   16160 bytes
  Size/request: 1616 bytes

Response time histogram:
  0.939 [1]     |■■■■■■■■■■■■■
  0.939 [1]     |■■■■■■■■■■■■■
  0.939 [0]     |
  0.939 [3]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.939 [0]     |
  0.939 [0]     |
  0.939 [2]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.940 [1]     |■■■■■■■■■■■■■
  0.940 [0]     |
  0.940 [1]     |■■■■■■■■■■■■■
  0.940 [1]     |■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9386 secs
  25% in 0.9389 secs
```

```
  50% in 0.9394 secs
  75% in 0.9398 secs
  90% in 0.9400 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3090 secs, 0.9386 secs, 0.9400 secs
  DNS-lookup:   0.0056 secs, 0.0054 secs, 0.0058 secs
  req write:    0.0004 secs, 0.0000 secs, 0.0012 secs
  resp wait:    0.6297 secs, 0.6286 secs, 0.6306 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0001 secs

Status code distribution:
  [200] 10 responses
```

100 requests

```
$ hey -n 100 -c 100 http://localhost:3000/materials

Summary:
  Total:        1.0364 secs
  Slowest:      1.0352 secs
  Fastest:      0.9700 secs
  Average:      1.0170 secs
  Requests/sec: 96.4892

  Total data:   161600 bytes
  Size/request: 1616 bytes

Response time histogram:
  0.970 [1]     |■
  0.977 [1]     |■
  0.983 [0]     |
  0.990 [4]     |■■■■■■
  0.996 [7]     |■■■■■■■■■■
  1.003 [3]     |■■■■
  1.009 [10]    |■■■■■■■■■■■■■■■
  1.016 [13]    |■■■■■■■■■■■■■■■■■■■■
  1.022 [13]    |■■■■■■■■■■■■■■■■■■■■
  1.029 [29]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.035 [19]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■

Latency distribution:
  10% in 0.9931 secs
  25% in 1.0090 secs
  50% in 1.0216 secs
  75% in 1.0269 secs
  90% in 1.0332 secs
  95% in 1.0344 secs
  99% in 1.0352 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3305 secs, 0.9700 secs, 1.0352 secs
  DNS-lookup:   0.0055 secs, 0.0040 secs, 0.0065 secs
  req write:    0.0016 secs, 0.0000 secs, 0.0103 secs
  resp wait:    0.6848 secs, 0.6403 secs, 0.7087 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0009 secs

Status code distribution:
  [200] 100 responses
```

1000 requests

```
$ hey -n 1000 -c 100 http://localhost:3000/materials

Summary:
  Total:        7.8970 secs
  Slowest:      1.0626 secs
  Fastest:      0.6127 secs
  Average:      0.7786 secs
  Requests/sec: 126.6300

  Total data:   1616000 bytes
  Size/request: 1616 bytes

Response time histogram:
  0.613 [1]     |
  0.658 [216]   |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.703 [122]   |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.748 [92]    |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.793 [92]    |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.838 [166]   |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.883 [174]   |∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
  0.928 [34]    |∎∎∎∎∎∎
  0.973 [17]    |∎∎∎
  1.018 [42]    |∎∎∎∎∎∎∎∎
```

```
  1.063 [44]    |∎∎∎∎∎∎∎∎


Latency distribution:
  10% in 0.6403 secs
  25% in 0.6733 secs
  50% in 0.7801 secs
  75% in 0.8549 secs
  90% in 0.9624 secs
  95% in 1.0102 secs
  99% in 1.0544 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0326 secs, 0.6127 secs, 1.0626 secs
  DNS-lookup:   0.0006 secs, 0.0000 secs, 0.0073 secs
  req write:    0.0002 secs, 0.0000 secs, 0.0217 secs
  resp wait:    0.7458 secs, 0.6126 secs, 0.9475 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0027 secs

Status code distribution:
  [200] 1000 responses
```

Conclusion:

For 10, 100, 1000 request each of request has average around 1s.

- **GET A MATERIAL BY ID**

10 request

```
$ hey -n 10 -c 10 http://localhost:3000/material/6432eae35f5f400947a051ac

Summary:
  Total:        0.9558 secs
  Slowest:      0.9554 secs
  Fastest:      0.9528 secs
  Average:      0.9540 secs
  Requests/sec: 10.4628

  Total data:   2330 bytes
  Size/request: 233 bytes

Response time histogram:
  0.953 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.953 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.953 [0]     |
  0.954 [2]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.954 [0]     |
  0.954 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.954 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.955 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.955 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.955 [1]     |■■■■■■■■■■■■■■■■■■■■
  0.955 [1]     |■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9529 secs
  25% in 0.9535 secs
  50% in 0.9542 secs
  75% in 0.9550 secs
  90% in 0.9554 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3210 secs, 0.9528 secs, 0.9554 secs
  DNS-lookup:   0.0046 secs, 0.0041 secs, 0.0049 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0006 secs
  resp wait:    0.6328 secs, 0.6314 secs, 0.6339 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0001 secs

Status code distribution:
  [200] 10 responses
```

100 request

```
$ hey -n 100 -c 100 http://localhost:3000/material/6432eae35f5f400947a051ac

Summary:
  Total:        1.0398 secs
  Slowest:      1.0394 secs
  Fastest:      0.9706 secs
  Average:      1.0185 secs
  Requests/sec: 96.1691

  Total data:   23300 bytes
  Size/request: 233 bytes

Response time histogram:
  0.971 [1]     |■
  0.977 [5]     |■■■■■■■
  0.984 [2]     |■■■
  0.991 [10]    |■■■■■■■■■■■■■■
  0.998 [0]     |
  1.005 [4]     |■■■■■■
  1.012 [6]     |■■■■■■■■
  1.019 [5]     |■■■■■■■
  1.026 [13]    |■■■■■■■■■■■■■■■■■■
```

```
  1.033 [25]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.039 [29]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9855 secs
  25% in 1.0080 secs
  50% in 1.0278 secs
  75% in 1.0333 secs
  90% in 1.0367 secs
  95% in 1.0382 secs
  99% in 1.0394 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3352 secs, 0.9706 secs, 1.0394 secs
  DNS-lookup:   0.0059 secs, 0.0037 secs, 0.0078 secs
  req write:    0.0010 secs, 0.0000 secs, 0.0035 secs
  resp wait:    0.6821 secs, 0.6402 secs, 0.7080 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0010 secs

Status code distribution:
  [200] 100 responses
```

1000 request

```
$ hey -n 1000 -c 100 http://localhost:3000/material/6432eae35f5f400947a051ac

Summary:
  Total:        7.2653 secs
  Slowest:      1.0654 secs
  Fastest:      0.6091 secs
  Average:      0.7092 secs
  Requests/sec: 137.6406

  Total data:   233000 bytes
  Size/request: 233 bytes

Response time histogram:
  0.609 [1]     |
  0.655 [313]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.700 [376]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.746 [174]   |■■■■■■■■■■■■■■■■■■■
  0.792 [34]    |■■■■
  0.837 [2]     |
  0.883 [0]     |
  0.928 [0]     |
  0.974 [7]     |■
  1.020 [25]    |■■■
  1.065 [68]    |■■■■■■■


Latency distribution:
  10% in 0.6315 secs
  25% in 0.6479 secs
  50% in 0.6719 secs
  75% in 0.7147 secs
  90% in 0.9560 secs
  95% in 1.0389 secs
  99% in 1.0571 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0329 secs, 0.6091 secs, 1.0654 secs
  DNS-lookup:   0.0010 secs, 0.0000 secs, 0.0138 secs
  req write:    0.0002 secs, 0.0000 secs, 0.0256 secs
  resp wait:    0.6760 secs, 0.6089 secs, 0.8085 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0034 secs

Status code distribution:
  [200] 1000 responses
```

Conclusion :

From request 10, 100, 1000 has same average performance in 1s.

- ## UPDATE A MATERIAL BY ID

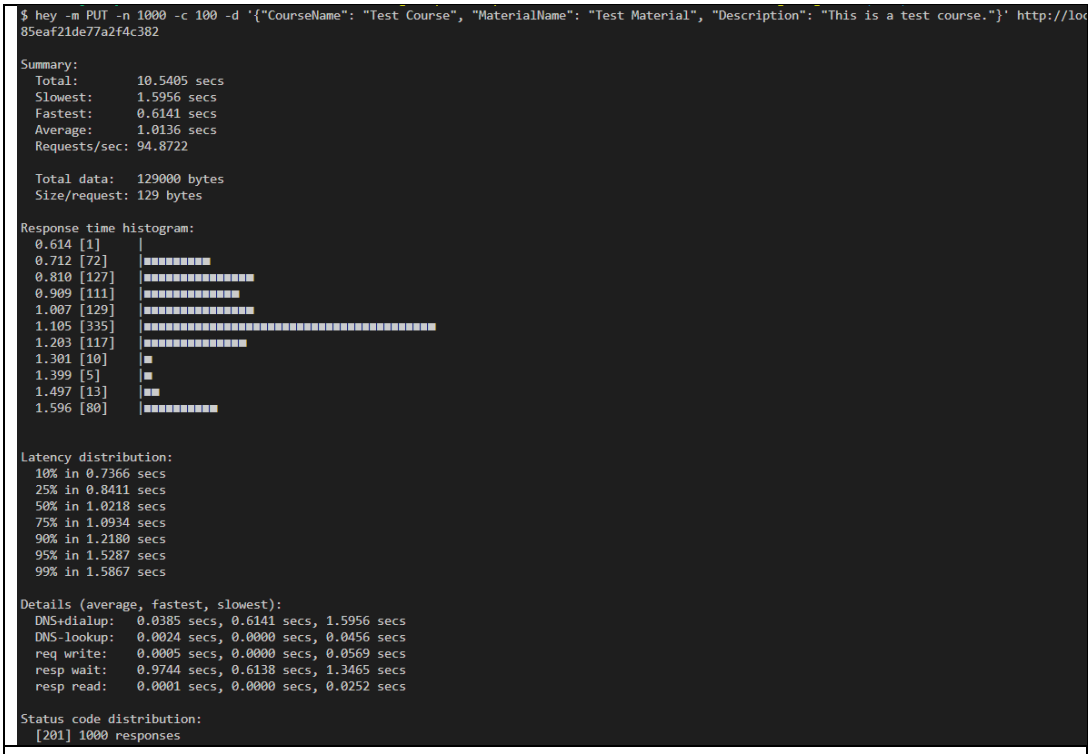
10 request

```
$ hey -m PUT -n 10 -c 10 -d '{"CourseName": "Test Course", "MaterialName": "Test Material", "Description": "This is a test course."}' http://localhos
af21de77a2f4c382

Summary:
  Total:        0.9423 secs
  Slowest:      0.9420 secs
  Fastest:      0.9320 secs
  Average:      0.9380 secs
  Requests/sec: 10.6125

  Total data:   1290 bytes
  Size/request: 129 bytes

Response time histogram:
  0.932 [1]     |■■■■■■■■■■■■■
  0.933 [1]     |■■■■■■■■■■■■■
  0.934 [1]     |■■■■■■■■■■■■■
  0.935 [0]     |
  0.936 [0]     |
  0.937 [1]     |■■■■■■■■■■■■■
  0.938 [0]     |
  0.939 [0]     |
  0.940 [1]     |■■■■■■■■■■■■■
  0.941 [2]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.942 [3]     |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 0.9320 secs
  25% in 0.9364 secs
  50% in 0.9406 secs
  75% in 0.9413 secs
  90% in 0.9420 secs
  0% in 0.0000 secs
  0% in 0.0000 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3114 secs, 0.9320 secs, 0.9420 secs
  DNS-lookup:   0.0077 secs, 0.0071 secs, 0.0088 secs
  req write:    0.0002 secs, 0.0000 secs, 0.0008 secs
  resp wait:    0.6263 secs, 0.6211 secs, 0.6304 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0002 secs

Status code distribution:
  [201] 10 responses
```

100 request

```
$ hey -m PUT -n 100 -c 100 -d '{"CourseName": "Test Course", "MaterialName": "Test Material", "Description": "This is a test course."}' http://localhos
5eaf21de77a2f4c382

Summary:
  Total:        1.3474 secs
  Slowest:      1.3468 secs
  Fastest:      1.0213 secs
  Average:      1.3120 secs
  Requests/sec: 74.2148

  Total data:   12900 bytes
  Size/request: 129 bytes

Response time histogram:
  1.021 [1]     |■
  1.054 [1]     |■
  1.086 [0]     |
  1.119 [0]     |
  1.151 [2]     |■
  1.184 [0]     |
  1.217 [0]     |
  1.249 [2]     |■
  1.282 [1]     |■
  1.314 [32]    |■■■■■■■■■■■■■■■■■■■■■
  1.347 [61]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


Latency distribution:
  10% in 1.2870 secs
  25% in 1.2939 secs
  50% in 1.3373 secs
  75% in 1.3426 secs
  90% in 1.3447 secs
  95% in 1.3460 secs
  99% in 1.3468 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.3731 secs, 1.0213 secs, 1.3468 secs
  DNS-lookup:   0.0187 secs, 0.0041 secs, 0.0406 secs
  req write:    0.0036 secs, 0.0000 secs, 0.0439 secs
  resp wait:    0.9352 secs, 0.6804 secs, 0.9982 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0003 secs

Status code distribution:
  [201] 100 responses
```

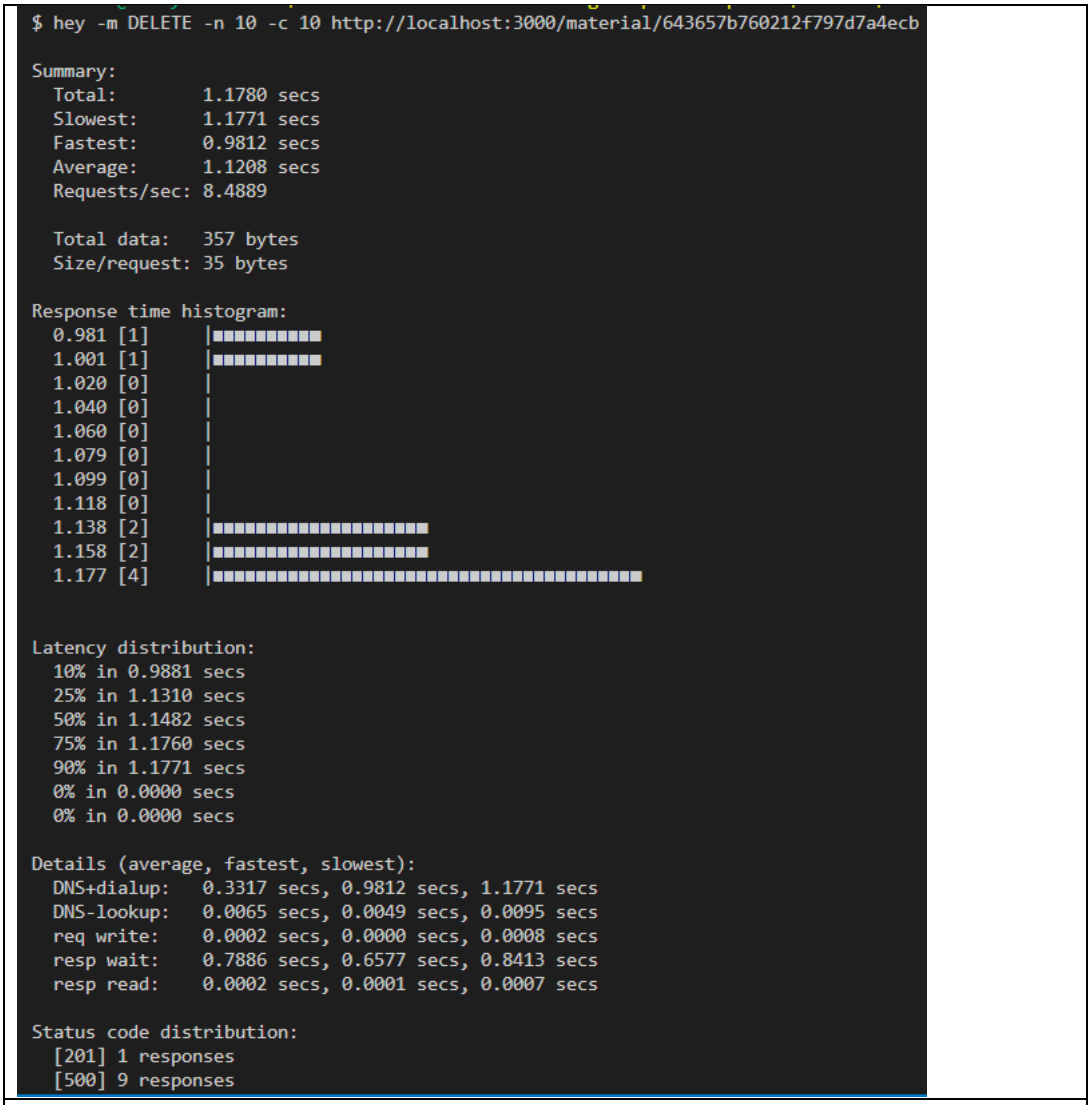
1000 request



Conclusion:
For 10, 100, 1000 request each of request has average around 1s.

- **DELETE A MATERIAL BY ID**

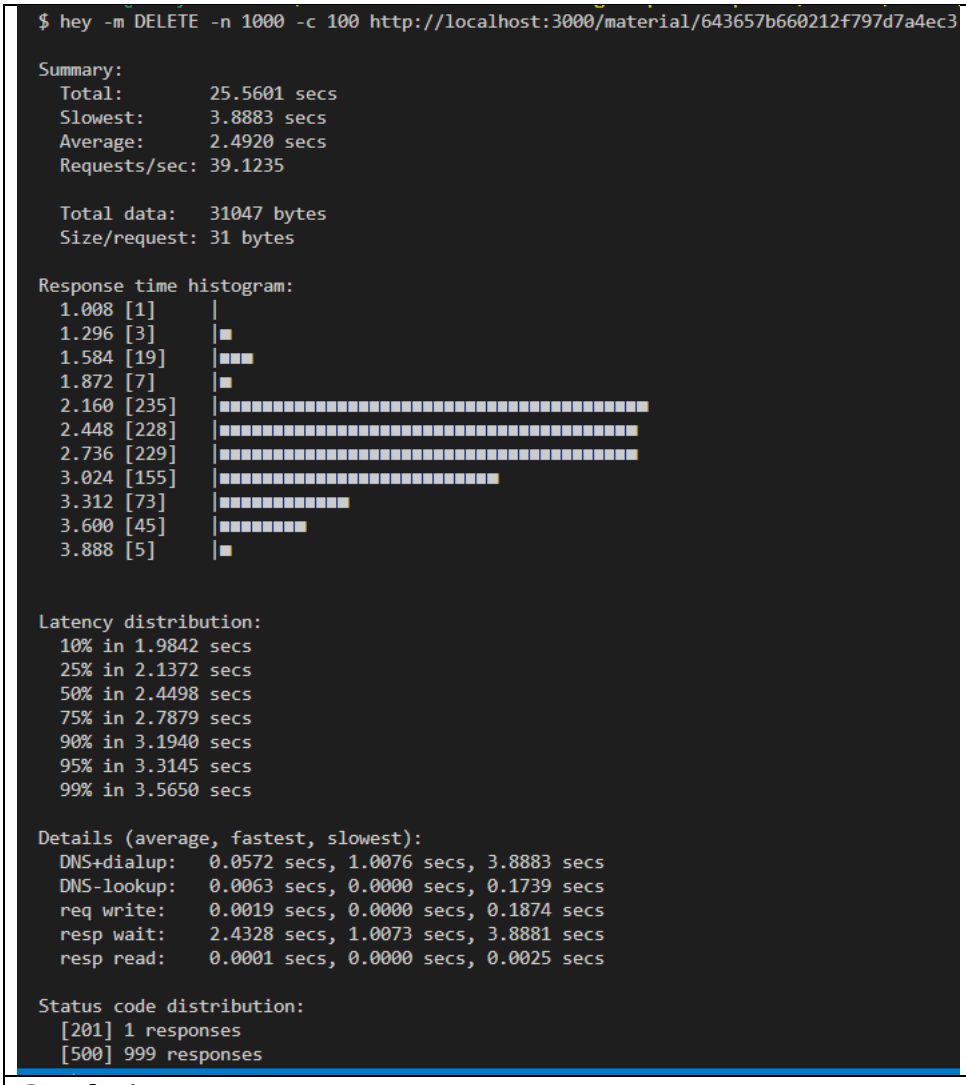
10 request


$ hey -m DELETE -n 10 -c 10 http://localhost:3000/material/643657b760212f797d7a4ecb
Summary:
Total: 1.1780 secs
Slowest: 1.1771 secs
Fastest: 0.9812 secs
Average: 1.1208 secs
Requests/sec: 8.4889
Total data: 357 bytes
Size/request: 35 bytes
Response time histogram:
0.981 [1]
1.001 [1]
1.020 [0]
1.040 [0]
1.060 [0]
1.079 [0]
1.099 [0]
1.118 [0]
1.138 [2]
1.158 [2]
1.177 [4]
Latency distribution:
10% in 0.9881 secs
25% in 1.1310 secs
50% in 1.1482 secs
75% in 1.1760 secs
90% in 1.1771 secs
0% in 0.0000 secs
0% in 0.0000 secs
Details (average, fastest, slowest):
DNS+dialup: 0.3317 secs, 0.9812 secs, 1.1771 secs
DNS-lookup: 0.0065 secs, 0.0049 secs, 0.0095 secs
req write: 0.0002 secs, 0.0000 secs, 0.0008 secs
resp wait: 0.7886 secs, 0.6577 secs, 0.8413 secs
resp read: 0.0002 secs, 0.0001 secs, 0.0007 secs
Status code distribution:
[201] 1 responses
[500] 9 responses

100 request

```
$ hey -m DELETE -n 100 -c 100 http://localhost:3000/material/6432ea758abc1630f7033c86

Summary:
  Total:        2.8130 secs
  Slowest:      2.8113 secs
  Fastest:      2.0120 secs
  Average:      2.6033 secs
  Requests/sec: 35.5487

  Total data:   3147 bytes
  Size/request: 31 bytes

Response time histogram:
  2.012 [1]     |
  2.092 [0]     |
  2.172 [0]     |
  2.252 [0]     |
  2.332 [0]     |
  2.412 [0]     |
  2.492 [0]     |
  2.571 [2]     |■
  2.651 [90]    |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  2.731 [0]     |
  2.811 [7]     |■■■


Latency distribution:
  10% in 2.5883 secs
  25% in 2.5933 secs
  50% in 2.5978 secs
  75% in 2.6016 secs
  90% in 2.6039 secs
  95% in 2.7921 secs
  99% in 2.8113 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.5532 secs, 2.0120 secs, 2.8113 secs
  DNS-lookup:   0.0204 secs, 0.0069 secs, 0.0387 secs
  req write:    0.0052 secs, 0.0000 secs, 0.2358 secs
  resp wait:    2.0448 secs, 1.6402 secs, 2.2167 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0005 secs

Status code distribution:
  [201] 1 responses
  [500] 99 responses
```

1000 request



Conclusion:
For 10, 100, 1000 request each of request has average around 2s.

E. Summary

The task performed by the group members Helsa Nesta Dhaifullah and Naily Khairiya was to create CRUD operations for courses and materials using Golang, Gin framework, and MongoDB. We established a connection between Go and MongoDB and created a database collection. In the course model, they defined five components: ID (primitive object), CourseName (string), Description (string), CreatedAt (time), and UpdatedAt (time). Similarly, they defined six components for the material model add materialName (string). For CRUD operations on courses, we created functions such as CreateCourse, GetAllCourses, ReadOneCourse, UpdateCourse, and DeleteCourse, and mapped them with different routes in main.go. The CreateMaterial, GetAllMaterial, GetOneMaterial, UpdateMaterial, and DeleteMaterial functions were used for CRUD operations on materials. Overall, the group successfully created and tested the CRUD features with the use hey library. Average performance all endpoint is 1s. So, the key value in hash map is work properly.