APPENDIX

1.MONOLITH CODE

IMPORT PACKAGE

```
1.    package main
2.
3.    import (
4.         "merch/config"
5.         "merch/model"
6.         "merch/usecase"
7.         "net/http"
8.
9.         "github.com/dgrijalva/jwt-go"
10.        "github.com/go-playground/validator/v10"
11.        "github.com/labstack/echo"
12.        "github.com/labstack/echo/middleware"
13.        _ "github.com/lib/pq"
14.   )
15.
```

CONNECT TO DB

```
1.    func DBCon() *sql.DB {
2.         db, err := sql.Open("postgres", "host=127.0.0.1 port=5433
user=postgres password=admin dbname=dbmerch sslmode=disable")
3.         if err != nil {
4.              log.Fatal(err)
5.         }
6.         // db.SetMaxOpenConns(0)
7.
8.         return db
```

REPOSITORY MERCHANT FUNCTION

```
1.    func   InsertMerchant(request   model.Merchant,   db   *sql.DB)
(model.Merchant, error) {
2.         var result model.Merchant
3.         query := `INSERT INTO tblmerchant (name)
4.         VALUES ($1)
5.         RETURNING idmerchant`
6.         err := db.QueryRow(query, request.Name).
7.              Scan(&result.Idmerchant)
8.         if err != nil {
9.              fmt.Println("Error Repo ", err)
10.             return result, err
11.        }
12.        return result, nil
13.
14.   }
15. func UpdateMerchant(request model.Merchant, db *sql.DB) (model.Merchant,
error) {
16.      var result model.Merchant
```

```
17.      ctx := context.Background()
18.
19.      tx, err := db.BeginTx(ctx, nil)
20.
21.      if err != nil {
22.           fmt.Println("Error Update - BeginTx ", err)
23.           tx.Rollback()
24.           return result, err
25.      }
26.
27.      res, err2 := tx.ExecContext(ctx, "UPDATE tblmerchant SET name=$1
   WHERE idmerchant=$2", request.Name, request.Idmerchant)
28.      if err2 != nil {
29.           fmt.Println("Error Update - ExecContext Update ", err)
30.           tx.Rollback()
31.           return result, err2
32.      }
33.      count, err3 := res.RowsAffected()
34.      if err3 != nil {
35.           fmt.Println("Error Update - RowsAffected Update ", err)
36.           tx.Rollback()
37.           return result, err3
38.      }
39.      if count == 0 {
40.           err = errors.New("error update")
41.           tx.Rollback()
42.           return result, err
43.      }
44.
45.      // sqlStatement := "UPDATE tblmerchant SET name=$1 WHERE id=$2"
46.      //     _,     err2     :=     db.Query(sqlStatement,     request.Name,
   request.Idmerchant)
47.      if err2 != nil {
48.           fmt.Println("Error Repo ", err)
49.           return result, err
50.      }
51.
52.      err = tx.Commit()
53.      if err != nil {
54.           fmt.Println("Error UpdateStokProduk - tx.Commit", err)
55.           return result, err
56.      }
57.      result.Idmerchant = request.Idmerchant
58.      return result, nil
59. }
```

REPOSITORY TRANSACTION FUNCTION

```
1. func InsertTransaction(request model.Transaction, db *sql.DB) (string,
   error) {
2.
3.     var result string
4.                query      :=      `INSERT      INTO      tbltransaction
   (transaction_code,transaction_date,payment_method,total)
```

```
5.      VALUES (CONCAT($1::text, LPAD(nextval('trx_no_counter')::text, 4,
   '0')),$2,$3,$4)
6.     RETURNING transaction_code`
7.            err     :=     db.QueryRow(query,     request.TransactionCode,
   request.TransactionDate,                                request.PaymentMethod,
   request.Total).Scan(&result)
8.
9.     if err != nil {
10.         fmt.Println("Error Repo ", err)
11.         return result, err
12.     }
13.
14.     return result, nil
15. }
16.
17. func  InsertTransactionDetail(request  []model.TransactionDetail,  db
   *sql.DB) (bool, error) {
18.
19.     vals := []interface{}{}
20.           query     :=     `INSERT     INTO     tbltransactiondetail
   (transaction_code,product_code,quantity,price)
21.     VALUES`
22.     for _, row := range request {
23.         query += "(?, ?, ?, ?),"
24.            vals  =  append(vals,  row.TransactionCode,  row.ProductCode,
   row.Quantity, row.Price)
25.     }
26.     query = query[0 : len(query)-1]
27.     query = ReplaceSQL(query, "?")
28.     stmt, _ := db.Prepare(query)
29.     _, err := stmt.Exec(vals...)
30.
31.     if err != nil {
32.         fmt.Println("Error Repo ", err)
33.         return false, err
34.     }
35.
36.     return true, nil
37.
38. }
39.
40. func    GetTransactionDetailsByCode(code    string,    db    *sql.DB)
   ([]model.TransactionDetail, error) {
41.
42.     var result []model.TransactionDetail
43.
44.     query := `SELECT id,transaction_code,product_code,quantity,price
45.     FROM tbltransactiondetail
46.     WHERE transaction_code=$1`
47.
48.     rows, err := db.Query(query, code)
49.     if err != nil {
50.         fmt.Println("Error Repo ", err)
51.         return result, err
52.     }
53.
```

```
54.     // defer rows.Close()
55.     for rows.Next() {
56.         var val model.TransactionDetail
57.                 err   :=   rows.Scan(&val.Id,   &val.TransactionCode,
   &val.ProductCode, &val.Quantity, &val.Price)
58.         if err != nil {
59.             fmt.Println("ERROR LOOP", err)
60.             return result, err
61.         }
62.         result = append(result, val)
63.
64.     }
65.
66.     return result, nil
67.
68. }
69.
70. func GetTransactionList(db *sql.DB) ([]model.Transaction, error) {
71.
72.     var result []model.Transaction
73.                             query                :=              `SELECT
   id,transaction_code,transaction_date,payment_method,total           from
   tbltransaction`
74.     rows, err := db.Query(query)
75.     if err != nil {
76.         fmt.Println("Error Repo ", err)
77.         return result, err
78.     }
79.     for rows.Next() {
80.         var val model.Transaction
81.                 err   :=   rows.Scan(&val.Id,   &val.TransactionCode,
   &val.TransactionDate, &val.PaymentMethod, &val.Total)
82.         if err != nil {
83.             fmt.Println("ERROR LOOP")
84.             return result, err
85.         }
86.         result = append(result, val)
87.
88.     }
89.     return result, nil
90. }
91.
92. func ReplaceSQL(old, searchPattern string) string {
93.     tmpCount := strings.Count(old, searchPattern)
94.     for m := 1; m <= tmpCount; m++ {
95.          old = strings.Replace(old, searchPattern, "$"+strconv.Itoa(m),
   1)
96.     }
97.     return old
98. }
99.
```

REPOSITORY USER

```
1. func InsertUser(request model.User, db *sql.DB) (model.User, error) {
2.  var result model.User
3.  query := `INSERT INTO tbluser (email,password)
4.  VALUES ($1,$2)
5.  RETURNING id`
6.  err := db.QueryRow(query, request.Email, request.Password).
7.       Scan(&result.Id)
8.  if err != nil {
9.       fmt.Println("Error Repo ", err)
10.           return result, err
11.      }
12.      return result, nil
13.
14. }
15.
16. func GetUserByEmail(email string, db *sql.DB) (model.User, error) {
17.      var result model.User
18.      err := db.QueryRow("SELECT id, email, password FROM tbluser where
    email = $1", email).Scan(&result.Id, &result.Email, &result.Password)
19.      if err != nil {
20.           fmt.Println("Error Repo ", err)
21.           return result, err
22.      }
23.      return result, nil
24.
25. }
26.
27. func GetAllUser(db *sql.DB) ([]model.User, error) {
28.
29.      var result []model.User
30.      rows, err := db.Query("SELECT  id, email, password FROM tbluser
    order by id")
31.      if err != nil {
32.           fmt.Println("Error Repo ", err)
33.           return result, err
34.      }
35.
36.      defer rows.Close()
37.      for rows.Next() {
38.           res := model.User{}
39.           err2 := rows.Scan(&res.Id, &res.Email, &res.Password)
40.           // // Exit if we get an error
41.           if err2 != nil {
42.                return result, err2
43.           }
44.           result = append(result, res)
45.      }
46.      return result, nil
47. }
```

USECASE MERCHANT

```
1. func InsertMerchant(request model.Merchant, db *sql.DB)
   model.ResponseMerchant {
2.  var result model.ResponseMerchant
3.  merchant, err := repo.InsertMerchant(request, db)
4.  if err != nil {
5.       fmt.Println("Err Usecase", err)
6.       result.Status = 401
7.       result.Desription = err.Error()
8.       return result
9.  }
10.      merchant.Name = request.Name
11.      result.Status = 200
12.      result.Desription = "Berhasil"
13.      result.Merchant = merchant
14.      return result
15. }
16. func GetMerchantById(id int, db *sql.DB) model.ResponseMerchant {
17.      var result model.ResponseMerchant
18.      merchant, err := repo.GetMerchantById(id, db)
19.      if err != nil {
20.           fmt.Println("Err Usecase", err)
21.           result.Status = 401
22.           result.Desription = err.Error()
23.           return result
24.      }
25.      result.Status = 200
26.      result.Desription = "Berhasil Delete"
27.      result.Merchant = merchant
28.      return result
29. }
30.
31. func UpdateMerchant(request model.Merchant, db *sql.DB)
   model.ResponseMerchant {
32.      var result model.ResponseMerchant
33.      single, err2 := repo.GetMerchantById(request.Idmerchant, db)
34.      if err2 != nil {
35.           fmt.Println("GET MU SALAH Err Usecase", err2)
36.           result.Status = 401
37.           result.Desription = "Id " + strconv.Itoa(single.Idmerchant) +
   "tidak ada " + err2.Error()
38.           return result
39.      }
40.      merchant, err := repo.UpdateMerchant(request, db)
41.      if err != nil {
42.           fmt.Println("UPDATE SALAH Err Usecase", err)
43.           result.Status = 401
44.           result.Desription = err.Error()
45.           return result
46.      }
47.      merchant.Name = request.Name
48.      result.Status = 200
49.      result.Desription = "Berhasil"
50.      result.Merchant = merchant
51.      return result
```

```
52. }
53.
54. func GetAllMerchant(db *sql.DB) model.ResponseMerchants {
55.      var result model.ResponseMerchants
56.      merchants, err := repo.GetAllMerchant(db)
57.      if err != nil {
58.           fmt.Println("Err Usecase", err)
59.           result.Status = 401
60.           result.Desription = err.Error()
61.           return result
62.      }
63.
64.      result.Status = 200
65.      result.Desription = "Berhasil"
66.      result.Merchants = merchants
67.      return result
68. }
```

USECASE TRANSACTION

```
1. func  InsertTransaction(request  model.RequestTransaction,  db  *sql.DB)
   model.ResponseAddTransaction {
2.  var result model.ResponseAddTransaction
3.  var trans model.Transaction
4.  var count int
5.  times := Timestamp()
6.  count = 0
7.
8.  timesufix := time.Now().Format("20060102150405")
9.  code := "TRX-" + timesufix + "-"
10.
11.      trans.TransactionCode = code
12.      trans.TransactionDate = times
13.      trans.Total = count
14.      trans.PaymentMethod = request.PaymentMethod
15.
16.      y, err := repo.InsertTransaction(trans, db)
17.      if err != nil {
18.           fmt.Println("Err Usecase", err)
19.           result.Status = 401
20.           result.Desription = err.Error()
21.           result.Success = false
22.           return result
23.      }
24.
25.      for i, x := range request.TransactionDetail {
26.           count = count + x.Price
27.           request.TransactionDetail[i].TransactionCode = y
28.      }
29.
30.      _,  err  =  repo.InsertTransactionDetail(request.TransactionDetail,
   db)
31.      if err != nil {
32.           fmt.Println("Err Usecase", err)
33.           result.Status = 401
34.           result.Desription = err.Error()
35.           result.Success = false
```

```
36.           return result
37.      }
38.
39.      result.Status = 200
40.      result.Desription = "Insert Success"
41.      result.Success = true
42.
43.      return result
44.
45. }
```

USECASE USER

```
1. func InsertUser(request model.User, db *sql.DB) model.ResponseUser {
2.  var result model.ResponseUser
3.
4.  request.Password, _ = HashPassword(request.Password)
5.
6.  users, err := repo.InsertUser(request, db)
7.  if err != nil {
8.       fmt.Println("Err Usecase", err)
9.       result.Status = 401
10.           result.Desription = err.Error()
11.           return result
12.      }
13.      users.Email = request.Email
14.      users.Password = request.Password
15.      result.Status = 200
16.      result.Desription = "Berhasil"
17.      result.User = users
18.      return result
19. }
20.
21. func     LoginUser(request     model.RequestLogin,     db     *sql.DB)
   model.ResponseLogin {
22.      var result model.ResponseLogin
23.
24.      users, err := repo.GetUserByEmail(request.Email, db)
25.      if err != nil {
26.           fmt.Println("Err Usecase", err)
27.           result.Status = 401
28.           result.Desription = err.Error()
29.           return result
30.      }
31.
32.      x := CheckPasswordHash(request.Password, users.Password)
33.
34.      if !x {
35.           fmt.Println("Wrong Password", err)
36.           result.Status = 401
37.           result.Desription = "Wrong Password"
38.           return result
39.      }
40.
41.      token, _ := GenerateToken(users.Id, users.Email)
42.      result.Status = 200
```

```
43.       result.Desription = "Berhasil"
44.       result.Email = users.Email
45.       result.Id = users.Id
46.       result.Token = token
47.
48.       return result
49.
50.  }
```

MICROSERVICE CODE

MERCHANT SERVER LAYER

```
1.  func NewHTTPServer(ctx context.Context, endpoints Endpoints) http.Handler
    {
2.   r := mux.NewRouter()
3.   r.Use(commonMiddleware)
4.
5.   r.Methods("POST").Path("/merchant/add").Handler(httptransport.NewServe
    r(
6.        endpoints.InsertMerchant,
7.        decodeMerchantReq,
8.        encodeResponse,
9.   ))
10.       r.Methods("POST").Path("/merchant/update").Handler(httptransport.N
    ewServer(
11.            endpoints.UpdateMerchant,
12.            decodeUpdateReq,
13.            encodeResponse,
14.       ))
15.
16.       r.Methods("GET").Path("/merchant/byid/{id}").Handler(httptransport
    .NewServer(
17.            endpoints.GetMerchantById,
18.            decodeIdReq,
19.            encodeResponse,
20.       ))
21.
22.       r.Methods("GET").Path("/merchant/list").Handler(httptransport.NewS
    erver(
23.            endpoints.GetAllMerchant,
24.            decodeGetReq,
25.            encodeResponse,
26.       ))
27.
28.       r.Methods("GET").Path("/merchant/delete/{id}").Handler(httptranspo
    rt.NewServer(
29.            endpoints.DeleteMerchant,
30.            decodeIdReq,
31.            encodeResponse,
32.       ))
33.
34.       return r
35.  }
36.
```

```
37. func commonMiddleware(next http.Handler) http.Handler {
38.       return      http.HandlerFunc(func(w      http.ResponseWriter,      r
    *http.Request) {
39.            w.Header().Add("Content-Type", "application/json")
40.            next.ServeHTTP(w, r)
41.       })
42. }
```

TRANSACTION SERVER LAYER

```
1. func NewHTTPServer(ctx context.Context, endpoints Endpoints) http.Handler
   {
2.  r := mux.NewRouter()
3.  r.Use(commonMiddleware)
4.
5.  r.Methods("POST").Path("/transaction/add").Handler(httptransport.NewSe
   rver(
6.       endpoints.InsertTransaction,
7.       decodeTransactionReq,
8.       encodeResponse,
9.  ))
10.
11.      r.Methods("GET").Path("/transaction/list").Handler(httptransport.N
   ewServer(
12.           endpoints.GetTransactionList,
13.           decodeGetReq,
14.           encodeResponse,
15.      ))
16.
17.      return r
18. }
19.
20. func commonMiddleware(next http.Handler) http.Handler {
21.      return      http.HandlerFunc(func(w      http.ResponseWriter,      r
   *http.Request) {
22.           w.Header().Add("Content-Type", "application/json")
23.           next.ServeHTTP(w, r)
24.      })
25. }
```

USER SERVER LAYER

```
1.
2. func NewHTTPServer(ctx context.Context, endpoints Endpoints) http.Handler
   {
3.  r := mux.NewRouter()
4.  r.Use(commonMiddleware)
5.
6.  r.Methods("POST").Path("/user/register").Handler(httptransport.NewServ
   er(
7.       endpoints.CreateUser,
8.       decodeUserReq,
9.       encodeResponse,
10.      ))
11.
12.      r.Methods("GET").Path("/user/byid/{id}").Handler(httptransport.New
   Server(
```

```
13.           endpoints.GetUser,
14.           decodeEmailReq,
15.           encodeResponse,
16.      ))
17.
18.      r.Methods("GET").Path("/user/list").Handler(httptransport.NewServe
   r(
19.           endpoints.GetManyUser,
20.           decodeGetReq,
21.           encodeResponse,
22.      ))
23.
24.      r.Methods("POST").Path("/user/login").Handler(httptransport.NewSer
   ver(
25.           endpoints.LoginUser,
26.           decodeUserReq,
27.           encodeResponse,
28.      ))
29.
30.      secure := r.PathPrefix("/user/auth").Subrouter()
31.      secure.Use(jwtVerify)
32.
33.      secure.Methods("GET").Path("/list").Handler(httptransport.NewServe
   r(
34.           endpoints.GetManyUser,
35.           decodeGetReq,
36.           encodeResponse,
37.      ))
38.
39.      return r
40. }
41.
42. func commonMiddleware(next http.Handler) http.Handler {
43.      return     http.HandlerFunc(func(w     http.ResponseWriter,     r
   *http.Request) {
44.           w.Header().Add("Content-Type", "application/json")
45.           next.ServeHTTP(w, r)
46.      })
47. }
48.
49. func jwtVerify(next http.Handler) http.Handler {
50.      return     http.HandlerFunc(func(w     http.ResponseWriter,     r
   *http.Request) {
51.           var JWT_SIGNING_METHOD = jwt.SigningMethodHS256
52.           var JWT_SIGNATURE_KEY = []byte("SKRIPSI")
53.           authorizationHeader := r.Header.Get("Authorization")
54.           if !strings.Contains(authorizationHeader, "Bearer") {
55.                http.Error(w, "Invalid token", http.StatusBadRequest)
56.                return
57.           }
58.
59.           tokenString := strings.Replace(authorizationHeader, "Bearer ",
   "", -1)
60.           token, err := jwt.Parse(tokenString, func(token *jwt.Token)
   (interface{}, error) {
```

```
61.                 if method, ok := token.Method.(*jwt.SigningMethodHMAC);
    !ok {
62.                     return nil, fmt.Errorf("Signing method invalid")
63.                 } else if method != JWT_SIGNING_METHOD {
64.                     return nil, fmt.Errorf("Signing method invalid")
65.                 }
66.
67.                 return JWT_SIGNATURE_KEY, nil
68.             })
69.             if err != nil {
70.                 http.Error(w, err.Error(), http.StatusBadRequest)
71.                 return
72.             }
73.
74.             claims, ok := token.Claims.(jwt.MapClaims)
75.             if !ok || !token.Valid {
76.                 http.Error(w, err.Error(), http.StatusBadRequest)
77.                 return
78.             }
79.             ctx := context.WithValue(context.Background(), "", claims)
80.             r = r.WithContext(ctx)
81.
82.             next.ServeHTTP(w, r)
83.
84.     })
85. }
```

USER STRUCT

```
1. type (
2.  CreateUserRequest struct {
3.       Email    string `json:"email"`
4.       Password string `json:"password"`
5.  }
6.  CreateUserResponse struct {
7.       Ok string `json:"ok"`
8.  }
9.  GetUserRequest struct {
10.          Id int `json:"id"`
11.     }
12.     GetLoginRequest struct {
13.          User User `json:"user"`
14.     }
15.     GetUserResponse struct {
16.          User User `json:"User"`
17.     }
18.     GetManyUserResponse struct {
19.          User []User `json:"User"`
20.     }
21.     LoginUserResponse struct {
22.          Response Response `json:"response"`
23.     }
24. )
25.
26. func encodeResponse(ctx context.Context, w http.ResponseWriter, response
    interface{}) error {
27.     return json.NewEncoder(w).Encode(response)
```

```
28. }
29.
30. func decodeUserReq(ctx context.Context, r *http.Request) (interface{},
    error) {
31.       var req CreateUserRequest
32.       err := json.NewDecoder(r.Body).Decode(&req)
33.       if err != nil {
34.            return nil, err
35.       }
36.       return req, nil
37. }
38.
39. func decodeGetReq(ctx context.Context, r *http.Request) (interface{},
    error) {
40.       return nil, nil
41. }
42.
43. func decodeEmailReq(ctx context.Context, r *http.Request) (interface{},
    error) {
44.       var req GetUserRequest
45.       vars := mux.Vars(r)
46.       ids := vars["id"]
47.       id, _ := strconv.Atoi(ids)
48.
49.       req = GetUserRequest{
50.            Id: id,
51.       }
52.       return req, nil
53. }
54.
55. func decodeLoginReq(ctx context.Context, r *http.Request) (interface{},
    error) {
56.       var req GetLoginRequest
57.       err := json.NewDecoder(r.Body).Decode(&req)
58.       if err != nil {
59.            return nil, err
60.       }
61.       fmt.Println(err)
62.       return req, nil
63. }
```

NGINX CONFIG FILE

```
1. server {
2.         listen       9090;
3.         server_name  localhost;
4.
5.         location /api/merchant/ {
6.          proxy_set_header X-Forwarded-For $remote_addr;
7.          proxy_set_header Host            $http_host;
8.          proxy_pass http://localhost:8081/merchant/;
9.         }
10.         location /api/user/ {
11.          proxy_set_header X-Forwarded-For $remote_addr;
12.          proxy_set_header Host            $http_host;
13.          proxy_pass http://localhost:8080/user/;
14.         }
```

```
15.         location /api/transaction/ {
16.          proxy_set_header X-Forwarded-For $remote_addr;
17.          proxy_set_header Host            $http_host;
18.          proxy_pass http://localhost:8082/transaction/;
19.         }
20.     }
```

FIRST DESIGN TESTING

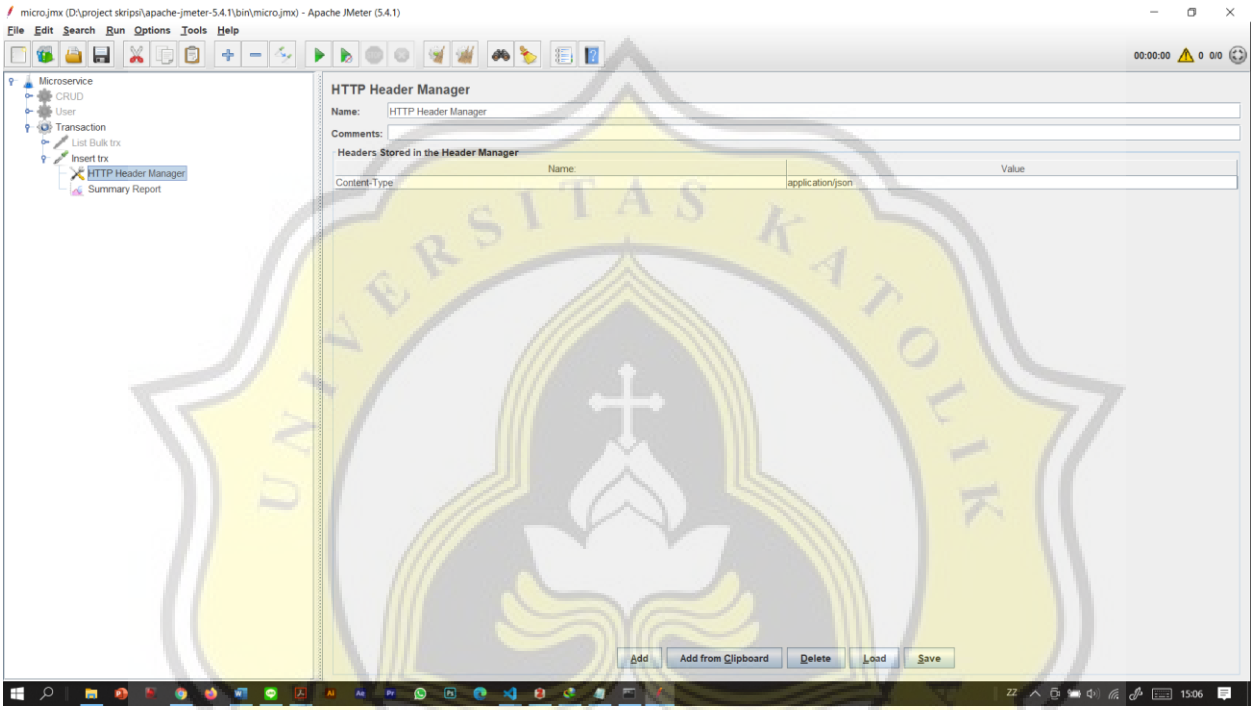

Figure 6.1 HTTP Header Manager JMeter

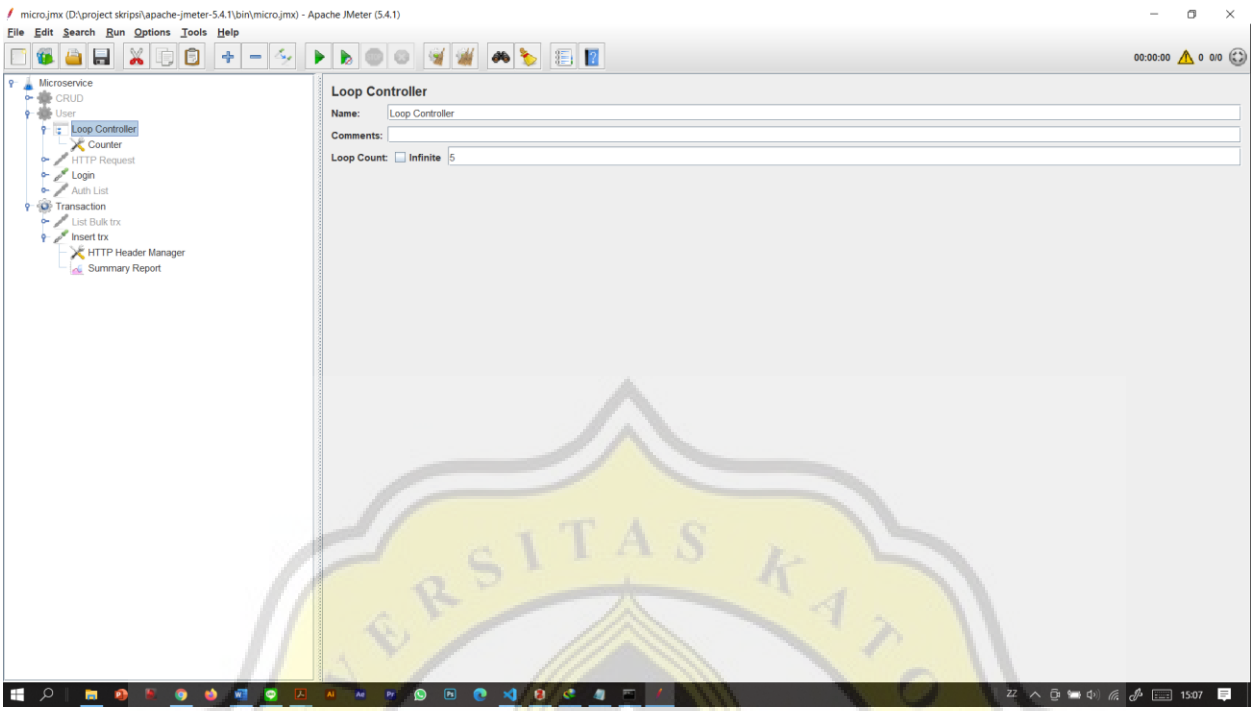

Figure 6.2 Loop Controller

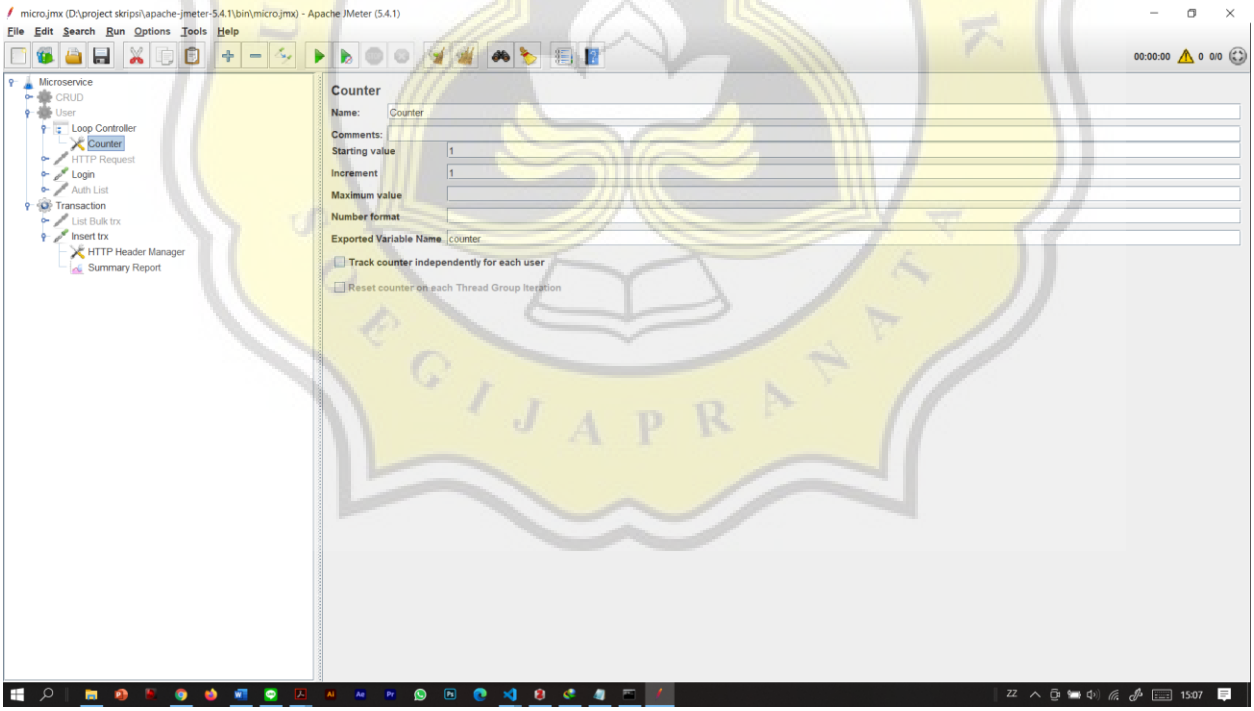

Figure 6.3 Counter

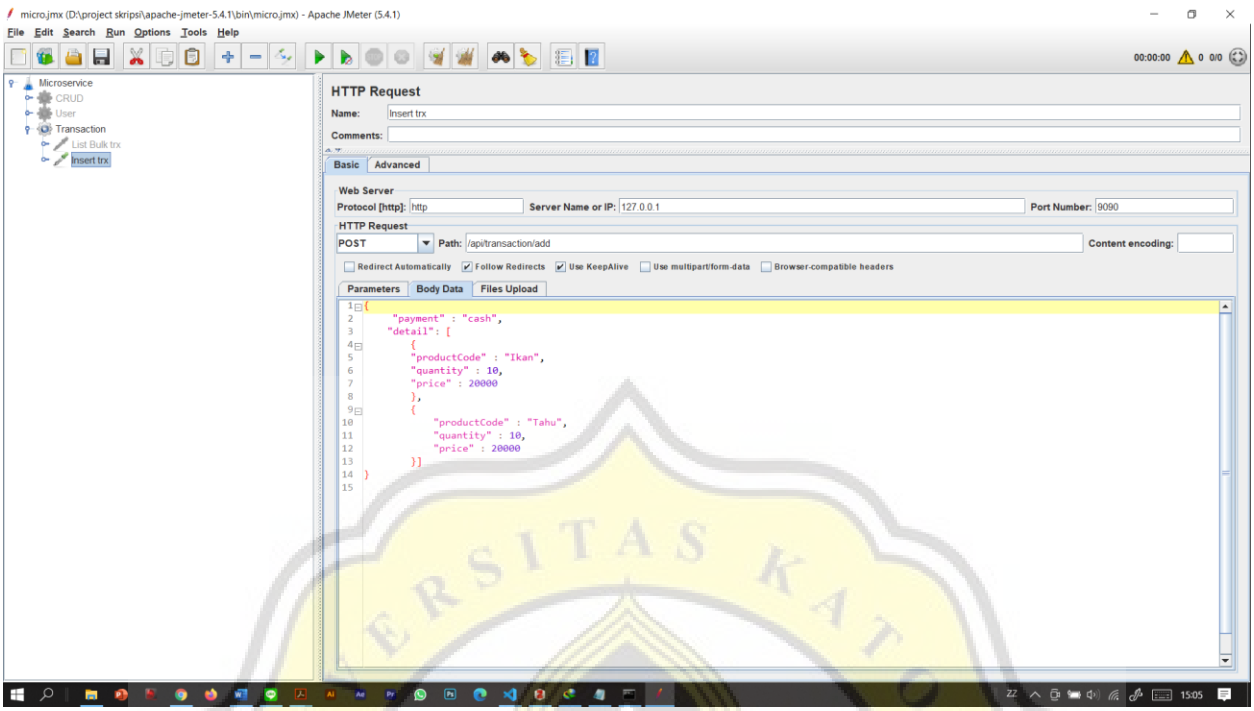

Figure 6.4 Transaction Testing

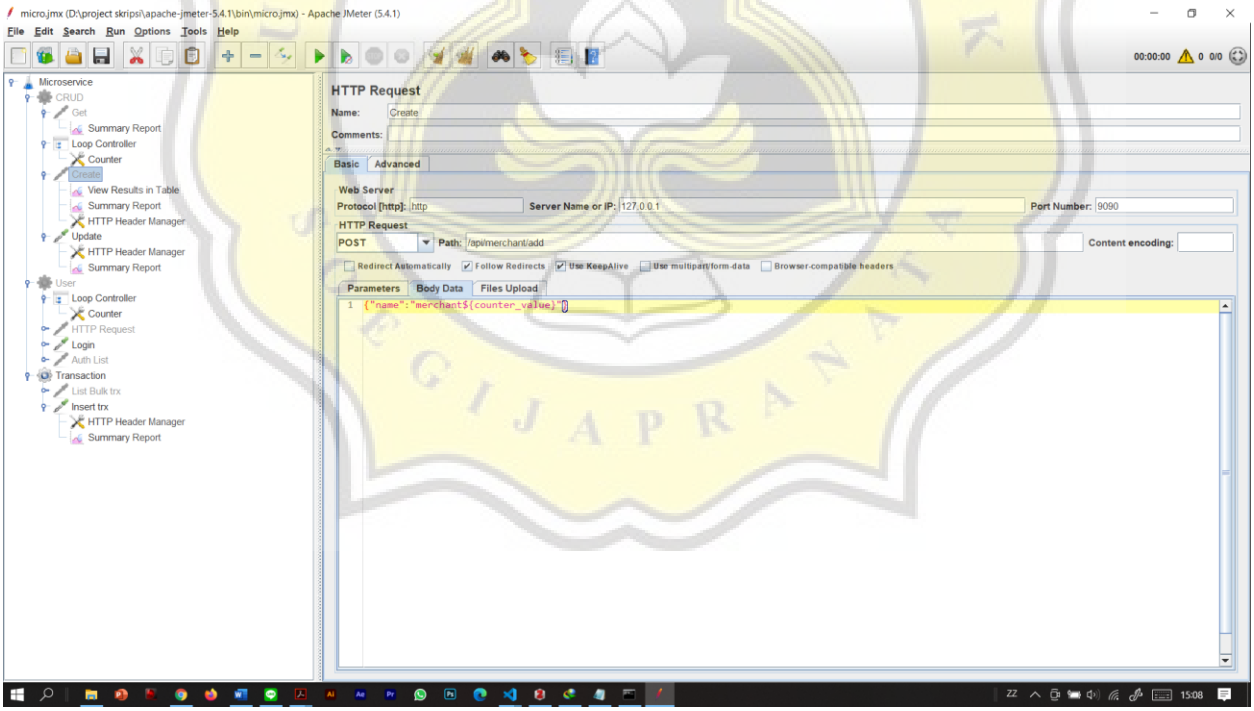

Figure 6.5 Merchant Testing

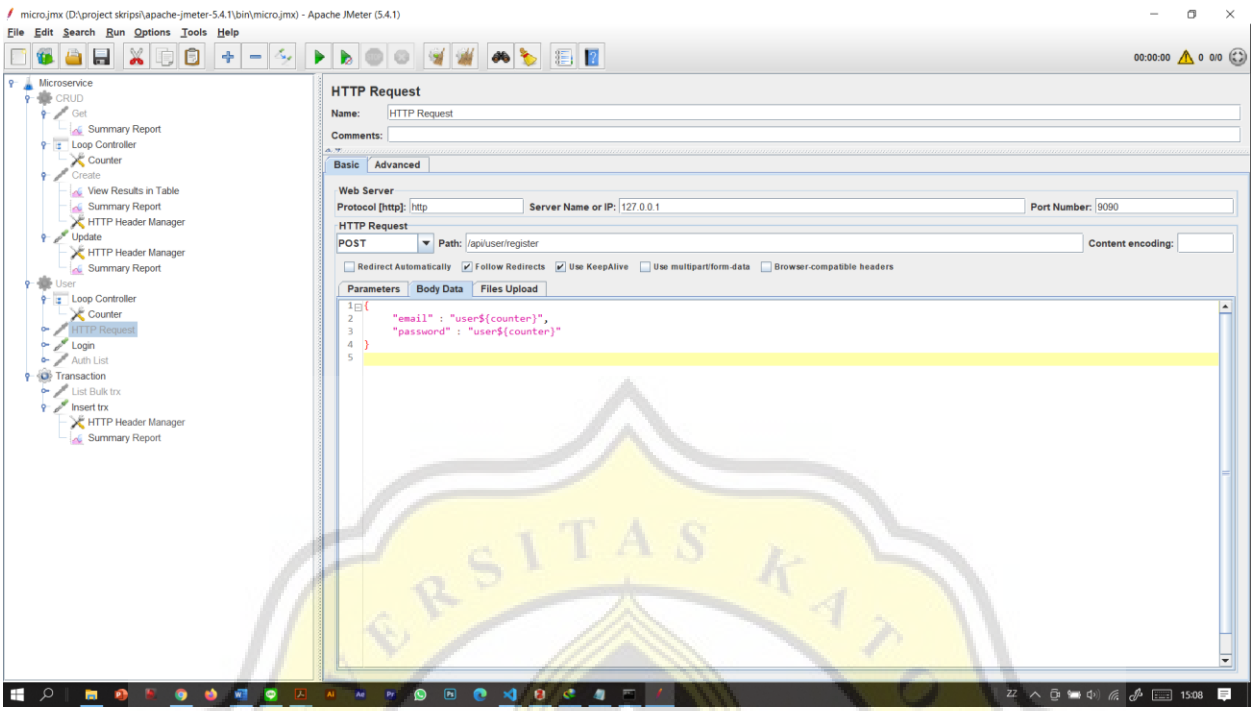

Figure 6.6 User Testing

FIRST DESIGN TESTING RESULT

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	100	4	3	15	1.87	0.00%	100.4/sec	27.26	12.65	278.0
TOTAL	100	4	3	15	1.87	0.00%	100.4/sec	27.26	12.65	278.0

Figure 6.7 Microservice 100 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	1000	2102	0	4792	1777.62	14.20%	117.0/sec	69.36	12.65	606.9
TOTAL	1000	2102	0	4792	1777.62	14.20%	117.0/sec	69.36	12.65	606.9

Figure 6.8 Microservice 1000 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	5000	1233	0	10164	2558.93	73.26%	202.9/sec	389.34	6.95	1964.8
TOTAL	5000	1233	0	10164	2558.93	73.26%	202.9/sec	389.34	6.95	1964.8

Figure 6.9 Microservice 5000 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	100	2	2	9	1.03	0.00%	101.9/sec	32.85	12.54	330.0
TOTAL	100	2	2	9	1.03	0.00%	101.9/sec	32.85	12.54	330.0

Figure 6.10 Monolith 100 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	1000	1288	0	2809	1085.51	13.70%	143.6/sec	89.79	15.25	640.3
TOTAL	1000	1288	0	2809	1085.51	13.70%	143.6/sec	89.79	15.25	640.3

Figure 6.11 Monolith 1000 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get	5000	641	0	5388	1121.50	75.60%	482.9/sec	963.14	14.50	2042.3
TOTAL	5000	641	0	5388	1121.50	75.60%	482.9/sec	963.14	14.50	2042.3

Figure 6.12 Monolith 5000 – Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	100	6	2	116	16.01	0.00%	100.6/sec	20.92	19.54	212.9
TOTAL	100	6	2	116	16.01	0.00%	100.6/sec	20.92	19.54	212.9

Figure 6.13 Monolith 100 - Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	1000	1688	0	3707	1573.99	34.80%	116.9/sec	119.03	14.88	1042.5
TOTAL	1000	1688	0	3707	1573.99	34.80%	116.9/sec	119.03	14.88	1042.5

Figure 6.14 Monolith 1000 – Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	5000	663	0	6534	1532.13	81.98%	322.2/sec	681.50	11.35	2166.1
TOTAL	5000	663	0	6534	1532.13	81.98%	322.2/sec	681.50	11.35	2166.1

Figure 6.15 Monolith 5000 – Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	100	5	3	101	9.98	0.00%	100.3/sec	16.75	19.78	171.0
TOTAL	100	5	3	101	9.98	0.00%	100.3/sec	16.75	19.78	171.0

Figure 6.16 Microservice 100 – Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	1000	2477	1	4498	1646.74	0.50%	95.4/sec	16.00	18.90	171.7
TOTAL	1000	2477	1	4498	1646.74	0.50%	95.4/sec	16.00	18.90	171.7

Figure 6.17 Microservice 1000 – Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Create	5000	2101	0	13004	3591.68	74.16%	194.0/sec	372.80	9.95	1968.1
TOTAL	5000	2101	0	13004	3591.68	74.16%	194.0/sec	372.80	9.95	1968.1

Figure 6.18 Microservice 5000 – Insert Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	100	4	3	15	1.92	0.00%	100.6/sec	16.80	22.98	171.0
TOTAL	100	4	3	15	1.92	0.00%	100.6/sec	16.80	22.98	171.0

Figure 6.19 Microservice 100 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	1000	2798	0	5584	2049.30	20.10%	97.7/sec	62.77	17.90	658.0
TOTAL	1000	2798	0	5584	2049.30	20.10%	97.7/sec	62.77	17.90	658.0

Figure 6.20 Microservice 1000 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	5000	1206	0	10825	2967.73	79.22%	294.2/sec	600.36	14.09	2089.3
TOTAL	5000	1206	0	10825	2967.73	79.22%	294.2/sec	600.36	14.09	2089.3

Figure 6.21 Microservice 5000 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	100	3	3	10	0.90	0.00%	100.5/sec	20.90	22.86	212.9
TOTAL	100	3	3	10	0.90	0.00%	100.5/sec	20.90	22.86	212.9

Figure 6.22 Monolith 100 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	1000	1211	0	5356	1624.82	46.00%	104.3/sec	133.40	12.86	1309.2
TOTAL	1000	1211	0	5356	1624.82	46.00%	104.3/sec	133.40	12.86	1309.2

Figure 6.23 Monolith 1000 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Update	5000	497	0	8255	1698.94	85.44%	302.1/sec	663.30	10.06	2248.3
TOTAL	5000	497	0	8255	1698.94	85.44%	302.1/sec	663.30	10.06	2248.3

Figure 6.24 Monolith 5000 – Update Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	27492	13189	35241	6916.75	0.00%	2.8/sec	0.74	0.61	272.9
TOTAL	100	27492	13189	35241	6916.75	0.00%	2.8/sec	0.74	0.61	272.9

Figure 6.25 Monolith 100 – Register User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1000	11512	0	83125	23605.69	76.80%	11.0/sec	22.02	0.56	2056.4
TOTAL	1000	11512	0	83125	23605.69	76.80%	11.0/sec	22.02	0.56	2056.4

Figure 6.26 Monolith 1000 – Register User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	29362	18205	38607	5757.63	0.00%	2.5/sec	0.42	0.58	171.0
TOTAL	100	29362	18205	38607	5757.63	0.00%	2.5/sec	0.42	0.58	171.0

Figure 6.27 Microservice 100 – Register User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1000	10526	0	60003	19823.79	86.40%	16.5/sec	11.65	3.08	724.4
TOTAL	1000	10526	0	60003	19823.79	86.40%	16.5/sec	11.65	3.08	724.4

Figure 6.28 Microservice 1000 – Register User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	100	28597	2519	41222	11016.41	0.00%	2.4/sec	0.85	0.53	369.0
TOTAL	100	28597	2519	41222	11016.41	0.00%	2.4/sec	0.85	0.53	369.0

Figure 6.29 Microservice 100 – Login Request

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	1000	10041	0	60006	19797.35	86.90%	16.4/sec	11.69	2.98	730.6
TOTAL	1000	10041	0	60006	19797.35	86.90%	16.4/sec	11.69	2.98	730.6

Figure 6.30 Microservice 1000 – Login Request

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	100	27702	2474	39590	10355.06	0.00%	2.5/sec	0.81	0.54	336.0
TOTAL	100	27702	2474	39590	10355.06	0.00%	2.5/sec	0.81	0.54	336.0

Figure 6.31 Monolith 100 – Login Request

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	1000	11742	0	88392	24307.93	77.50%	11.2/sec	22.83	0.55	2086.7
TOTAL	1000	11742	0	88392	24307.93	77.50%	11.2/sec	22.83	0.55	2086.7

Figure 6.32 Monolith 1000 – Login Request

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	100	98	5	402	126.40	0.00%	80.5/sec	14.46	35.59	184.0
TOTAL	100	98	5	402	126.40	0.00%	80.5/sec	14.46	35.59	184.0

Figure 6.33 Monolith 100 – Insert Trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	1000	1790	0	6494	2346.14	49.60%	100.9/sec	135.95	22.49	1379.9
TOTAL	1000	1790	0	6494	2346.14	49.60%	100.9/sec	135.95	22.49	1379.9

Figure 6.34 Monolith 1000 – Insert Trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	5000	2134	0	12431	3837.42	69.18%	163.6/sec	295.79	22.30	1851.9
TOTAL	5000	2134	0	12431	3837.42	69.18%	163.6/sec	295.79	22.30	1851.9

Figure 6.35 Monolith 5000 – Insert Trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	100	6	5	33	3.52	0.00%	100.5/sec	15.51	44.76	158.0
TOTAL	100	6	5	33	3.52	0.00%	100.5/sec	15.51	44.76	158.0

Figure 6.36 Microservice 100 – Insert Trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	1000	1849	0	5739	1737.77	29.90%	76.0/sec	65.76	23.72	886.4
TOTAL	1000	1849	0	5739	1737.77	29.90%	76.0/sec	65.76	23.72	886.4

Figure 6.37 Microservice 1000 – Insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Insert trx	5000	739	0	9863	1867.47	86.06%	338.3/sec	744.73	21.00	2254.4
TOTAL	5000	739	0	9863	1867.47	86.06%	338.3/sec	744.73	21.00	2254.4

Figure 6.38 Microservice 5000 – Insert trx

DESIGN 2 TESTING RESULT

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	100	12	7	31	5.00	0.00%	100.1/sec	32.26	12.61	330.0
TOTAL	100	12	7	31	5.00	0.00%	100.1/sec	32.26	12.61	330.0

Figure 6.39 100 monolith - Get merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	1000	4698	31	6650	1880.22	27.30%	142.7/sec	132.36	13.07	950.0
TOTAL	1000	4698	31	6650	1880.22	27.30%	142.7/sec	132.36	13.07	950.0

Figure 6.40 1000 monolith - Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	5000	4073	25	11115	2663.80	66.30%	359.2/sec	643.96	15.25	1835.8
TOTAL	5000	4073	25	11115	2663.80	66.30%	359.2/sec	643.96	15.25	1835.8

Figure 6.41 5000 monolith - Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	100	15	8	106	13.76	0.00%	97.4/sec	33.28	12.55	350.0
TOTAL	100	15	8	106	13.76	0.00%	97.4/sec	33.28	12.55	350.0

Figure 6.42 100 microservice - Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	1000	2343	23	5965	2161.40	47.20%	161.9/sec	93.66	18.62	592.2
TOTAL	1000	2343	23	5965	2161.40	47.20%	161.9/sec	93.66	18.62	592.2

Figure 6.43 1000 microservice - Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	5000	3823	87	10970	2275.67	89.96%	423.0/sec	371.74	40.78	899.8
TOTAL	5000	3823	87	10970	2275.67	89.96%	423.0/sec	371.74	40.78	899.8

Figure 6.44 5000 microservice - Get Merchant

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	100	27339	2542	42521	10482.49	0.00%	2.3/sec	0.83	0.52	369.0
TOTAL	100	27339	2542	42521	10482.49	0.00%	2.3/sec	0.83	0.52	369.0

Figure 6.45 100 microservice - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	1000	10126	39	60151	19480.48	86.50%	16.6/sec	22.84	1.97	1411.7
TOTAL	1000	10126	39	60151	19480.48	86.50%	16.6/sec	22.84	1.97	1411.7

Figure 6.46 1000 microservice - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	100	29629	11484	37844	7082.00	0.00%	2.6/sec	0.84	0.57	336.0
TOTAL	100	29629	11484	37844	7082.00	0.00%	2.6/sec	0.84	0.57	336.0

Figure 6.47 100 monolith - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	1000	13115	2041	88281	23463.36	77.70%	11.3/sec	23.12	0.56	2095.9
TOTAL	1000	13115	2041	88281	23463.36	77.70%	11.3/sec	23.12	0.56	2095.9

Figure 6.48 1000 monolith - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	100	13	9	41	4.78	0.00%	99.8/sec	17.93	44.44	184.0
TOTAL	100	13	9	41	4.78	0.00%	99.8/sec	17.93	44.44	184.0

Figure 6.49 100 monolith - Insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	1000	5092	13	11870	3871.89	47.70%	80.8/sec	105.48	18.82	1336.9
TOTAL	1000	5092	13	11870	3871.89	47.70%	80.8/sec	105.48	18.82	1336.9

Figure 6.50 1000 monolith - Insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	5000	7649	121	22260	5279.70	76.48%	220.3/sec	437.40	23.07	2033.1
TOTAL	5000	7649	121	22260	5279.70	76.48%	220.3/sec	437.40	23.07	2033.1

Figure 6.51 5000 monolith - Insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	100	21	11	123	20.68	0.00%	99.5/sec	15.35	44.60	158.0
TOTAL	100	21	11	123	20.68	0.00%	99.5/sec	15.35	44.60	158.0

Figure 6.52 100 microservice - Insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	1000	2304	22	6869	2497.20	65.50%	144.1/sec	186.21	34.94	1323.5
TOTAL	1000	2304	22	6869	2497.20	65.50%	144.1/sec	186.21	34.94	1323.5

Figure 6.53 1000 microservice - insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	5000	6523	45	29908	5848.40	79.86%	165.8/sec	338.33	16.30	2090.0
TOTAL	5000	6523	45	29908	5848.40	79.86%	165.8/sec	338.33	16.30	2090.0

Figure 6.54 5000 microservice - insert trx

DESIGN 3 TESTING RESULT

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	100	134	14	1100	302.98	0.00%	89.4/sec	14.57	40.06	167.0
TOTAL	100	134	14	1100	302.98	0.00%	89.4/sec	14.57	40.06	167.0

Figure 6.55 100 Design 3 - insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	1000	4269	27	8066	3494.69	42.30%	122.7/sec	28.37	55.00	236.8
TOTAL	1000	4269	27	8066	3494.69	42.30%	122.7/sec	28.37	55.00	236.8

Figure 6.56 1000 Design 3 - insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Insert trx	5000	9439	723	27895	6206.70	71.72%	178.9/sec	65.69	77.05	376.0
TOTAL	5000	9439	723	27895	6206.70	71.72%	178.9/sec	65.69	77.05	376.0

Figure 6.57 5000 Design 3 - insert trx

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	100	25002	10742	39539	9340.17	0.00%	2.5/sec	0.91	0.55	378.0
TOTAL	100	25002	10742	39539	9340.17	0.00%	2.5/sec	0.91	0.55	378.0

Figure 6.58 100 Design 3 - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Login	1000	11084	312	60209	20824.84	86.80%	16.5/sec	5.49	3.72	339.7
TOTAL	1000	11084	312	60209	20824.84	86.80%	16.5/sec	5.49	3.72	339.7

Figure 6.59 1000 Design 3 - Login User

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	100	24	12	135	20.25	0.00%	99.4/sec	34.85	12.81	359.0
TOTAL	100	24	12	135	20.25	0.00%	99.4/sec	34.85	12.81	359.0

Figure 6.60 100 Design 3 - Merchant Get

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	1000	2320	31	4861	1242.61	0.70%	202.2/sec	70.85	26.06	358.8
TOTAL	1000	2320	31	4861	1242.61	0.70%	202.2/sec	70.85	26.06	358.8

Figure 6.61 1000 Design 3 - Merchant Get

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
Get	5000	7511	75	17187	4770.78	58.26%	288.8/sec	100.24	37.23	355.4
TOTAL	5000	7511	75	17187	4770.78	58.26%	288.8/sec	100.24	37.23	355.4

Figure 6.62 5000 Design 3 - Merchant Get

PLAGIARISM CHECK.ORG

0.59% PLAGIARISM APPROXIMATELY

0.05% IN QUOTES

Report #14119645

INTRODUCTION Background In this modern era, the requirement for digitalized systems increasing rapidly. With the Covid-19 Pandemic happening, many major living aspects tend to be online. Government and companies attempt to cope with the change of lifestyle by providing the online system and or application. This leads developers to create a high-performance system that is capable to serve the demand. One of the most common answers used by developers is web application. Web applications demanded to be agile, fast, and capable of handling such traffic. When it comes to building or developing an online system, developers need to choose what kind of architecture will be used for the system. When facing the challenge, a developer needs the most suitable architecture that most fit the case whether uses microservices or monolithic architecture. Both architectures offer different benefits. Microservice recently become popular because many large companies start migrating from monolith to microservices but on the other hand. many organizations are