СОДЕРЖАНИЕ

1 Реализация программного обеспечения 5
1.1 Развертывание k8s кластера 5
1.2 Реализация idP сервиса 11
1.3 Реализация клиента к idP 15
1.4 Тестирование программного обеспечения 17

2 Исследование характеристик программного обеспечения . . 20
2.1 Описание проводимого исследования 20
2.2 Технические характеристики устройства 22
2.3 Полученные результаты 23

ЗАКЛЮЧЕНИЕ 24

ВВЕДЕНИЕ

Целью дипломной практики является реализация программного-алгоритмического комплекса выпускной квалификационной работы на тему *"система авторизации инфраструктурных сервисов"*, а также проведение функционального тестирование и исследование характеристик реализованного программного обеспечения.

Задачи практики:

1) реализовать программно-алгоритмический комплекс выпускной квалификационной работы,

2) выполнить функциональное тестирование и предоставить пример,

3) провести исследование выполнения запросов между инфраструктурными сервисами с включенной и выключенной авторизацией.

1 Реализация программного обеспечения

Основные средства реализации:

1) k3d — утилита для поднятия k8s кластера локально, использует docker,
2) kubectl — утилита для ручного просмотра логов и состояния k8s кластера,
3) docker — инструмент для контейнеризации приложений. Используется в реализации для создания sidecar контейнеров,
4) ghcr.io — используется для загрузки docker образов в k8s кластер,
5) Golang — язык программирования, в основном использующийся для написания приложений в микросервисной архитектуре.

На рисунке 1.1 приведена структура реализованного проекта.

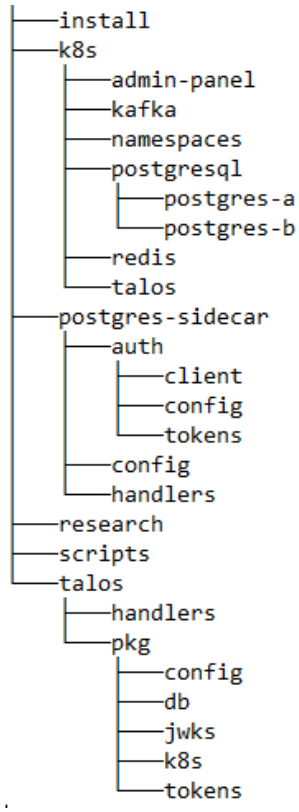

Рисунок 1.1 – Структура проекта

1.1 Развертывание k8s кластера

В листинге 1.1 приведен скрипт развертывания k8s сервисов.

Листинг 1.1 – Скрипт развертывания k8s кластера

```
#!/bin/bash

k3d cluster create bmstucluster \
  --api-port 6443 \
  --servers-memory 4G \
  --agents-memory 4G \
  --k3s-arg
     "--kubelet-arg=eviction-hard=memory.available<500Mi@server:*"
     \
  --k3s-arg
     "--kubelet-arg=eviction-hard=memory.available<500Mi@agent:*"
     \
  --k3s-arg "--kubelet-arg=image-gc-high-threshold=90@server:*" \
  --k3s-arg "--kubelet-arg=image-gc-low-threshold=80@server:*" \
  --k3s-arg "--kubelet-arg=fail-swap-on=false@server:*" \
  --kubeconfig-update-default \
  --k3s-arg "--kube-apiserver-arg=service-account-jwks-uri= \
        https://kubernetes.default.svc/openid/v1/jwks@server:*" \
  --k3s-arg "--kube-apiserver-arg=service-account-issuer= \
        https://kubernetes.default.svc@server:*"

# talos
docker build -t ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
   ./talos
docker push ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
k3d image import ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
   -c bmstucluster --keep-tools

# run sidecar code in sidecar containter:
docker build -t
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest
   ./postgres-sidecar
docker push
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest
k3d image import
   ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:latest -c
   bmstucluster --keep-tools

kubectl apply -f k8s/namespaces/
# kubectl apply -k k8s/namespaces/
```

```
namespaces=("postgres-a" "postgres-b" "talos")
for ns in "${namespaces[@]}"; do
  if ! kubectl get secret ghcr-secret -n "$ns" >/dev/null 2>&1;
     then
    kubectl create secret docker-registry ghcr-secret \
      --docker-server=ghcr.io \
      --docker-username=perpetua1g0d \
      --docker-password="$GH_PAT" \
      --namespace="$ns"
    echo "Secret GHCR created in namespace: $ns"
  else
    echo "Secret already exists in namespace: $ns"
  fi
done

kubectl apply -f k8s/talos/
kubectl apply -f k8s/postgresql/postgres-a/
kubectl apply -f k8s/postgresql/postgres-b/
```

В листингах 1.2 и 1.3 приведен пример конфигурации сервиса вместе с сайдкаром в одном поде, а также конфигурация развертывания сервиса idP.

Листинг 1.2 – Конфигурация развертывания PostgreSQL сервиса с сайдкаром

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-a
  namespace: postgres-a
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-a
  template:
    metadata:
      labels:
        app: postgres-a
    spec:
      serviceAccountName: default
      imagePullSecrets:
        - name: ghcr-secret
```

```
containers:
  - name: postgres
    image: postgres:13-alpine
    env:
      - name: POSTGRES_INITDB_ARGS
        value: "--data-checksums"
      - name: POSTGRES_PASSWORD
        value: "password"
      - name: POSTGRES_USER
        value: "admin"
      - name: POSTGRES_DB
        value: "appdb"
      - name: POSTGRES_PORT
        value: "5434"
    ports:
      - containerPort: 5434
    volumeMounts:
      - name: postgresql-data
        mountPath: /var/lib/postgresql/data
      - name: config
        mountPath: /etc/postgresql/postgresql.conf
        subPath: postgresql.conf
      - name: init-script
        subPath: init.sql
        mountPath: /docker-entrypoint-initdb.d/init.sql
      - name: shared-env
        mountPath: /etc/postgres-env
    lifecycle:
      postStart:
        exec:
          command:
            - "/bin/sh"
            - "-c"
            - |
              echo $POSTGRES_USER >
                /etc/postgres-env/POSTGRES_USER
              echo $POSTGRES_PASSWORD >
                /etc/postgres-env/POSTGRES_PASSWORD
              echo $POSTGRES_DB >
                /etc/postgres-env/POSTGRES_DB
              echo $POSTGRES_HOST >
```

```
                  /etc/postgres-env/POSTGRES_HOST
                echo $POSTGRES_PORT >
                  /etc/postgres-env/POSTGRES_PORT
      resources:
        limits:
          memory: "256Mi"
          cpu: "250m"

    - name: sidecar
      image:
        ghcr.io/perpetua1g0d/bmstu-diploma/postgres-sidecar:lates
      volumeMounts:
        - name: shared-env
          mountPath: /etc/postgres-env
      env:
        - name: SERVICE_NAME
          value: "postgres-a"
        - name: SIGN_AUTH_ENABLED
          value: "false"
        - name: VERIFY_AUTH_ENABLED
          value: "false"
        - name: INIT_TARGET_SERVICE
          value: "postgres-b"
        - name: RUN_BENCHMARKS_ON_INIT
          value: "true"

        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      ports:
        - containerPort: 8080
      resources:
        limits:
          memory: "1G"
          cpu: "5"

  volumes:
    - name: postgresql-data
      emptyDir: {}
    - name: config
```

```
          configMap:
            name: postgresql-config
        - name: init-script
          configMap:
            name: postgres-init-script
        - name: shared-env
          emptyDir: {}
```

Листинг 1.3 – Конфигурация развертывания сервиса idP

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: talos
  namespace: talos
spec:
  replicas: 1
  selector:
    matchLabels:
      app: talos
  template:
    metadata:
      labels:
        app: talos
    spec:
      serviceAccountName: default
      imagePullSecrets:
        - name: ghcr-secret
      containers:
        - name: talos
          image:
      ghcr.io/perpetua1g0d/bmstu-diploma/talos:latest
          ports:
            - containerPort: 8080
          resources:
            limits:
              memory: "128Mi"
              cpu: "100m"
```

1.2 Реализация idP сервиса

idP сервис был реализован на языке программирования Golang и получил k8s кластере имя *talos*.

В листинге 1.4 приведена SQL схема infra2infra и таблица для хранения прав.

Листинг 1.4 – Схема и таблица для хранения прав

```
CREATE SCHEMA IF NOT EXISTS infra2infra;

CREATE TABLE infra2infra."Permissions" (
    ClientName TEXT NOT NULL,
    ServerName TEXT NOT NULL,
    roles TEXT[] NOT NULL
);
```

В листингах 1.5–1.7 приведена реализация OIDC обработчиков HTTP запросов к сервису idP. Обработчики слушают HTTP запросы на получение сертификатов по следующим путям:

1) */realms/infra2infra/.well-known/openid-configuration* — обработчик запросов на получение OIDC конфигурации,

2) */realms/infra2infra/protocol/openid-connect/token* — обработчик запросов на выпуск и получение токена idP,

3) */realms/infra2infra/protocol/openid-connect/certs* — обработчик запросов на получение сертификатов idP.

Листинг 1.5 – Реализация обработчика запросов на сертификаты

```
func CertsHandler(keys *jwks.KeyPair) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        jwks := keys.JWKS()
        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(jwks)
    }
}

func (k *KeyPair) JWKS() jose.JSONWebKeySet {
    jwk := jose.JSONWebKey{
        Key:          k.PrivateKey.Public(),
```

```
        Certificates: []*x509.Certificate{k.Certificate},
        KeyID:        k.KeyID,
        Algorithm:    "RS256",
        Use:          "sig",
    }

    return jose.JSONWebKeySet{Keys: []jose.JSONWebKey{jwk}}
}
```

Листинг 1.6 – Реализация обработчика запросов на OIDC конфигурацию

```
func OpenIDConfigHandler(cfg *config.Config) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        tokenEndpointPath :=
           "/realms/infra2infra/protocol/openid-connect/token"
        certsEndpointPath :=
           "/realms/infra2infra/protocol/openid-connect/certs"
        response := map[string]interface{}{
            "issuer":                                  cfg.Issuer,
            "token_endpoint":                          cfg.Issuer
               + tokenEndpointPath,
            "jwks_uri":                                cfg.Issuer
               + certsEndpointPath,
            "grant_types_supported":
               []string{grantTypeTokenExchange},
            "id_token_signing_alg_values_supported":
               []string{"RS256"},
        }

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(response)
    }
}
```

Листинг 1.7 – Реализация обработчика запросов на выпуск токена

```
const (
    grantTypeTokenExchange =
       "urn:ietf:params:oauth:grant-type:token-exchange" // RFC
       8693
    k8sTokenType           =
       "urn:ietf:params:oauth:token-type:jwt:kubernetes"
)
```

```
type TokenRequest struct {
    GrantType        string `form:"grant_type"`
    SubjectTokenType string `form:"subject_token_type"`
    SubjectToken     string `form:"subject_token"`
    Scope            string `form:"scope"`
}

func NewTokenHandler(ctx context.Context, cfg *config.Config,
   keys *jwks.KeyPair) (http.HandlerFunc, error) {
    issuer, err := NewIssuer(cfg, keys)
    if err != nil {
        return nil, fmt.Errorf("failed to create issued: %w",
           err)
    }

    k8sVerifier, err := k8s.NewVerifier(ctx)
    if err != nil {
        return nil, fmt.Errorf("failed to create k8s verifier:
           %w", err)
    }

    return func(w http.ResponseWriter, r *http.Request) {
        if err := r.ParseForm(); err != nil {
            log.Printf("failed to parse form request params:
               %v", err)
            http.Error(w, `{"error":"invalid_request"}`,
               http.StatusBadRequest)
            return
        }

        log.Printf("Incoming request: Method=%s, URL=%s,
           Body=%s", r.Method, r.URL, r.Form)

        req := TokenRequest{
            GrantType:        r.FormValue("grant_type"),
            SubjectTokenType: r.FormValue("subject_token_type"),
            SubjectToken:     r.FormValue("subject_token"),
            Scope:            r.FormValue("scope"),
        }

        if req.GrantType != grantTypeTokenExchange {
```

```
            log.Printf("unexpected grant_type: %s",
               req.GrantType)
            http.Error(w, `{"error":"unsupported_grant_type"}`,
               http.StatusBadRequest)
            return
        } else if req.SubjectTokenType != k8sTokenType {
            log.Printf("unexpected subject_token_type: %s",
               req.GrantType)
            http.Error(w,
               `{"error":"unsupported_subject_token_type"}`,
               http.StatusBadRequest)
            return
        }

        clientID, _, err :=
           k8sVerifier.VerifyWithClient(req.SubjectToken)
        if err != nil {
            log.Printf("failed to verify k8s token: %v", err)
            http.Error(w, `{"error":"token_not_verified"}`,
               http.StatusBadRequest)
            return
        }

        issueResp, err := issuer.IssueToken(clientID, req.Scope)
        if err != nil {
            log.Printf("failed to issue talos token: %v", err)
            http.Error(w, `{"error":"access_denied"}`,
               http.StatusForbidden)
            return
        }

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(issueResp)

        log.Printf("token issued, clientID: %s, scope: %s",
           clientID, req.Scope)
    }, nil
}
```

1.3 Реализация клиента к idP

Был реализован клиент для получения публичного сертификата idP и фонового получения токена для проверки токена входящего запроса.

Пример использования клиента к idP для проверки токена из входящего HTTP запроса в инфраструктурном сервисе приведен на листинге 1.8

Листинг 1.8 – Проверка токена входящего запроса

```
type QueryRequest struct {
    SQL    string `json:"sql"`
    Params []any  `json:"params"`
}

func NewQueryHandler(ctx context.Context, authClient
   *auth_client.AuthClient) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        log.Printf("Incoming request: %s %s", r.Method, r.URL)

        cfg := config.GetConfig()

        if cfg.VerifyAuthEnabled {
            token := r.Header.Get("X-I2I-Token")
            if token == "" {
                respondError(w, "missing token",
                   http.StatusUnauthorized)
                return
            }

            requiredRole := "RO"
            sqlQuery := strings.ToUpper(r.URL.Query().Get("sql")
            if !strings.Contains(sqlQuery), "SELECT") {
                requiredRole = "RW"
            }

            if verifyErr := authClient.VerifyToken(token,
               []string{requiredRole}); verifyErr != nil {
                log.Printf("failed to verify token: %v",
                   verifyErr)
                respondError(w, "forbidden: token has no
                   required roles", http.StatusUnauthorized)
                return
            }
```

```
        log.Printf("successfully verified incoming token")
    }

    db, err := sql.Open("postgres", fmt.Sprintf(
        "host=%s port=%s user=%s password=%s dbname=%s
           sslmode=disable",
        cfg.PostgresHost,
        cfg.PostgresPort,
        cfg.PostgresUser,
        cfg.PostgresPassword,
        cfg.PostgresDB,
    ))
    if err != nil {
        respondError(w, "database connection failed",
           http.StatusInternalServerError)
        return
    }
    defer db.Close()

    var req QueryRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err !=
       nil {
        respondError(w, fmt.Sprintf("invalid request: %v",
           err), http.StatusBadRequest)
        return
    }

    start := time.Now()
    rows, err := db.Query(req.SQL, req.Params...)
    if err != nil {
        respondError(w, fmt.Sprintf("query failed: %v",
           err), http.StatusBadRequest)
        return
    }
    defer rows.Close()

    json.NewEncoder(w).Encode(map[string]interface{}{
        "status":  "success",
        "latency": time.Since(start).String(),
    })
```

```
    }
}
```

1.4 Тестирование программного обеспечения

Для функционального тестирования были написаны unit-тесты. Пример unit теста, реализованного с использованием Arrange-Act-Assert паттерна, для проверки k8s JWT токена приведен в листинге 1.9.

Листинг 1.9 – Тест для проверки k8s токена

```
func Test_Verify(t *testing.T) {
    // Arrange
    token := "eyJhbGciOiJSUzI1NiIsImtpZCI6Ilp2UFNpYlpISzk1YlhGRjh
        KSjJlY25MWHZRZ29aV25lMXB3UV9IYUh5TmsifQ.eyJhdWQiOlsiaHR0c
        HM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiLC
        JrM3MiXSwiZXhwIjoxNzc5NjUxMTU2LCJpYXQiOjE3NDgxMTUxNTYsIml
        zcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2YyIsImp0aSI6
        ImM0ZWUzNWFmLTFmZWYtNDNmYi1iNDYyLTJiNjk0OTM5MTdjMCIsImt1Y
        mVybmV0ZXMuaW8iOnsibmFtZXNwYWNlIjoicG9zdGdyZXMtYSIsIm5vZG
        UiOnsibmFtZSI6ImszZC1ibXN0dWNsdXN0ZXItc2VydmVyLTAiLCJ1aWQ
        iOiI3YzEwM2NlOC05OTA2LTQ3NWMtOGM5Ni1jNGZiZjIyOWM1YTAifSwi
        cG9kIjp7Im5hbWUiOiJwb3N0Z3Jlcy1hLTY3OTRmY2I1ZjctcWI5em0iL
        CJ1aWQiOiI5MTE3NDllNy01NTFiLTQ2M2UtYmNmZC03ZjEyNGFhZTgxNW
        UifSwic2VydmljZWFjY291bnQiOnsibmFtZSI6ImRlZmF1bHQiLCJ1aWQ
        iOiIyNDhkNDE3Mi0xZDdlLTRiMWEtYmRlNS05NzAyN2FiNDFlMmQifSwi
        d2FybmFmdGVyIjoxNzQ4MTE4NzYzfSwibmJmIjoxNzQ4MTE1MTU2LCJzd
        WIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6cG9zdGdyZXMtYTpkZWZhdW
        x0In0.spCWGajmAAENHgK5zG5NTX2dz82S0gunARu9-ncvurmV5XqEKOK
        SEypC9Rm3ap2WfwOei0zm4-0Hbi3fdYZUZkSDZO-HLSHxPZcQoGvVjiKD
        OAIRDGwNb8bNVqkPPyy1Q8K5cD1anJwPiXYFzIQ4ZKDu_Ikp5ajkA3KYp
        ZUmPLFQ3a09k8ycTpvdnjSzCfIdEaUiO4jFlGJWEYKGo9XuY0VVjyyjwJ
        AdDOj6Ry0aIJzJzQTjS-IUs_dL7XVQUSlxp4mZvhLhnrhiL6uU59tX1QV
        tliZ3MgkO3XN_F2G5kuoigFN0NQp8EpiCeC2-e9T-rchZ3MR8KbPp8lgs
        4SA1iA"

    jwk := JWK{
        N: `xItwcttR4qVTD4bBfsUDgpICFnoBk1H8qyN3jSVemH1wlPyn6CLn2
        aUmjQHW25f2LcraZr1_t7l0ogmaR46Gn7uyYGBEtIsNnjvvoAUVbmd8vI
        hPJI9flzDjJys4CEjefo1YFooD4YfqDei0GEYG2TYy42mO3TR6O3--47P
        bLIyZ2cbmHTwU-t_apqc3NUs0Sd6_gjDb0hrX0cFlOvBfL-J-3XEe4Zxe
        w7-qDnjQGIKdSEgS-v-wCwr30iqK9yDfHO9cHUtRNirLb4dybeOh3_vBM
```

```
        MLCVpKtH6GonDEVyRv7qJCigEinpHB78Uq0PAb_l8SOHougk2qp8-Cp3n
        b7rw`,
        E: "AQAB",
    }

    wantAud := jwt.ClaimStrings{
        "https://kubernetes.default.svc.cluster.local",
        "k3s",
    }

    publicKey, err := makeRSAPublicKey(jwk)
    if err != nil {
        t.Fatalf("failed to create public rsa key: %v", err)
    }

    verifier := &Verifier{
        publicKey: publicKey,
    }

    // Act
    gotClientID, gotClaims, gotErr :=
       verifier.VerifyWithClient(token)

    // Assert
    if gotErr != nil {
        t.Errorf("failed to verify token: %v", gotErr)
    } else if gotClientID != testClientID {
        t.Errorf("expected clientID: %s, got: %s", testClientID,
           gotClientID)
    }

    gotAud, gotAudErr := gotClaims.GetAudience()
    if gotAudErr != nil {
        t.Errorf("got unexpected aud err: %v", gotAudErr)
    }
    assert.EqualValues(t, wantAud, gotAud)
}
```

Вывод

В данном разделе были описаны средства реализации программного-алгоритмического комплекса, способы развертывания k8s кластера, в том числе сервисов с сайдкаром, приведена реализация idP сервиса, а также пример тестирования сервиса idP.

2 Исследование характеристик программного обеспечения

В исследовании было проведено сравнение времени выполнения запроса с включенной и выключенной авторизацией от одного инфраструктурного сервиса к другому.

2.1 Описание проводимого исследования

На листингах 2.1–2.2 приведена реализация запроса к инфраструктурному сервису, а также запуск выполнения фиксированного количества запросов параллельно.

Листинг 2.1 – Реализация запроса к PostgreSQL сервису

```
func sendBenchmarkQuery(cfg *config.Config, authClient
   *auth_client.AuthClient) {
    target :=
       fmt.Sprintf("http://%s.%s.svc.cluster.local:8080%s",
        cfg.InitTarget,
        cfg.InitTarget,
        cfg.ServiceEndpoint,
    )

    reqBody, _ := json.Marshal(map[string]interface{}{
        "sql":    `INSERT INTO log (message) VALUES ($1)`,
        "params": []interface{}{fmt.Sprintf("Write from %s, ts:
           %s", cfg.Namespace, time.Now())},
    })

    req, err := http.NewRequest("POST", target,
       bytes.NewBuffer(reqBody))
    if err != nil {
        log.Fatalf("failed to create post request: %v", err)
        return
    }

    if cfg.SignAuthEnabled {
        token, err := authClient.Token(cfg.InitTarget)
        if err != nil {
            log.Fatalf("failed to issue token in auth client on
               scope %s: %v", cfg.InitTarget, err)
```

```
            return
        }
        req.Header.Set("X-I2I-Token", token)
    }
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{Timeout: 4 * time.Second}
    resp, err := client.Do(req)

    errMsg := handlers.RespErr{}
    var respBytes []byte
    if resp != nil && resp.Body != nil {
        respBytes, _ = io.ReadAll(resp.Body)
        _ = json.Unmarshal(respBytes, &errMsg)
    }

    if err != nil {
        log.Fatalf("Initial query failed: %v; errMsg: %s", err,
           errMsg.Error)
        return
    }
    defer resp.Body.Close()
}
```

Листинг 2.2 – Реализация запуска выполнения параллельных запросов

```
func runBenchmarks(cfg *config.Config, authClient
   *auth_client.AuthClient) {
    file, err := os.Create(benchmarksResultsFile)
    if err != nil {
        log.Fatalf("Cannot create results file: %v", err)
    }
    defer file.Close()
    writer := csv.NewWriter(file)
    defer writer.Flush()
    writer.Write([]string{"requests", "time_ms", "operation",
       "sign_enabled", "sign_disabled"})

    requestCount := []int64{100, 250, 500, 750, 1000}
    rerunCount := 10
    for _, reqCount := range requestCount {
        var avgTime float64 = 0
        for _ = range rerunCount {
```

```
            wg := &sync.WaitGroup{}
            wg.Add(int(reqCount))

            start := time.Now()
            for i := 0; i < int(reqCount); i++ {
                go func() {
                    defer wg.Done()
                    sendBenchmarkQuery(cfg, authClient)
                }()
            }
            wg.Wait()

            duration := time.Since(start).Milliseconds()
            avgTime += float64(duration)
        }

        avgTime = avgTime / float64(rerunCount*int(reqCount))
        log.Printf("finished %d requests, avg: %f", reqCount,
           avgTime)
        writer.Write([]string{
            strconv.FormatInt(reqCount, 10),
            strconv.FormatFloat(avgTime, 'f', 2, 64),
            "write",
            fmt.Sprintf("%v", cfg.SignAuthEnabled),
            fmt.Sprintf("%v", cfg.VerifyAuthEnabled),
        })
    }
}
```

2.2 Технические характеристики устройства

Технические характеристики устройства, на котором проводилось исследование:

1) процессор Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz,

2) оперативная память 8 ГБ,

3) операционная система Ubuntu 21.0.

Исследование проводилось на ноутбуке. Во время исследования ноутбук не был нагружен посторонними приложениями, которые не относятся к

исследованию, а также ноутбук был подключен к сети питания.

2.3 Полученные результаты

Исследование проводилось при включенной и выключенной авторизации 100, 250, 500, 750, 1000 параллельных запросов из одного инфраструктурного сервиса к другому. Результаты для каждого количества запросов были усреднены путем запуска 10 раз.

Графики полученных усредненных результатов представлены на рисунке 2.1.

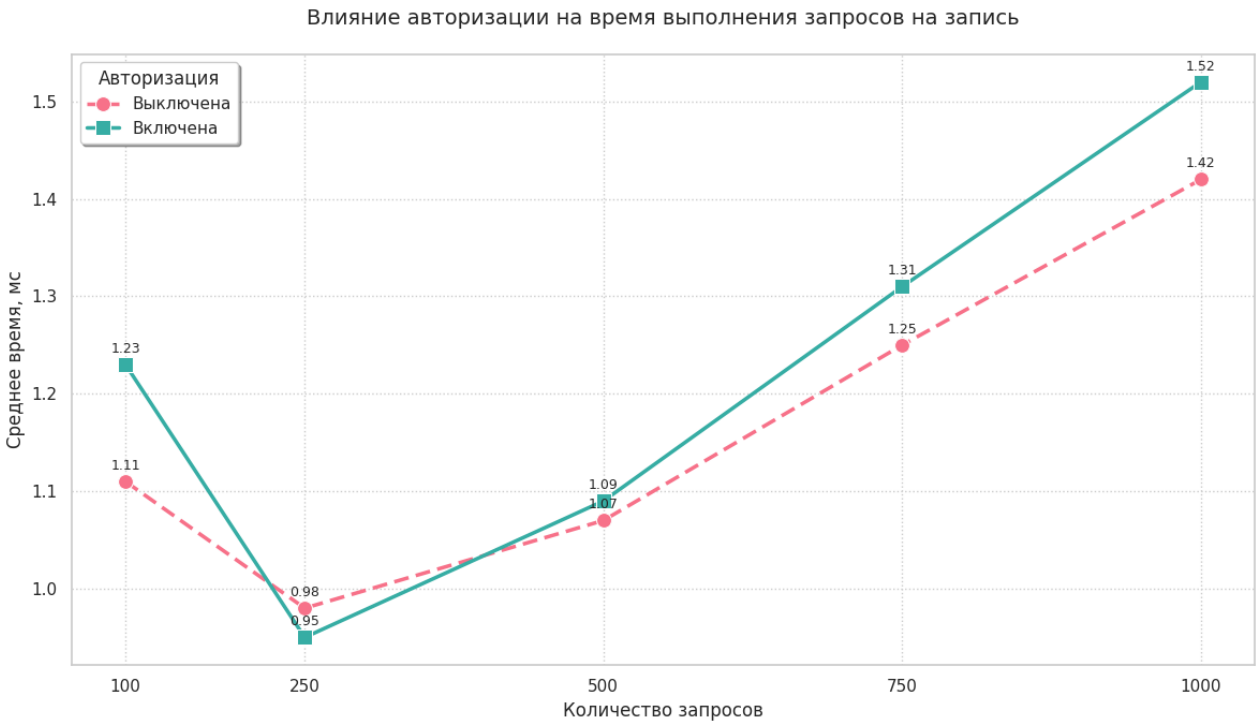

Рисунок 2.1 – Результаты исследования

Вывод

Как видно по графикам, время выполнения запросов с включенной авторизацией оказались в среднем на 10% дольше времени выполнения запросов с выключенной авторизацией. Это не окажет существенного влияния на работу системы из инфраструктурных сервисов.

ЗАКЛЮЧЕНИЕ

В рамках практики был реализован программного-алгоритмический комплекс *"система авторизации инфраструктурных сервисов"*, а также проведение функционального тестирование и исследование характеристик реализованного программного обеспечения.

Были выполнены следующие задачи практики:

1) реализовать программно-алгоритмический комплекс выпускной квалификационной работы,

2) выполнить функциональное тестирование и предоставить пример,

3) провести исследование выполнения запросов между инфраструктурными сервисами с включенной и выключенной авторизацией.