

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

к лабораторной работе №2

По курсу: «Тестирование и отладка ПО»

Тема: «Интеграционное и E2E тестирование»

Студент ИУ7-75Б Афимин И. Е.

Преподаватель Жаров С. В.

Результаты выполнения ЛР

1) Запуск CI/CD в GitHub Actions

```
1 name: Simple Build
 2 on:
 3 push:
    branches: [master]
 5 pull request:
    branches: [master]
8 jobs:
9 tests:
     runs-on: ubuntu-latest
11
     services:
                                  -- Поднимаю сервер Постгреса
12
      postgres:
13
        image: postgres:12
14
        env:
15
          POSTGRES USER: mdb
          POSTGRES PASSWORD: mdb
         POSTGRES_DB: mdb
17
18
        ports:
          - 5432:5432
19
20
        options: >-
21
          --health-cmd pg isready
           --health-interval 10s
23
           --health-timeout 5s
24
           --health-retries 5
25
26
    steps:
27
      - name: Install Go
28
        uses: actions/setup-go@v2
29
         with:
30
          go-version: 1.16.x
31
32
      - name: Checkout code
        uses: actions/checkout@v2
34
35
       - name: Create DataBase
                                -- Запускаю скрипт создания таблиц в БД
36
        env:
37
          DB HOST: localhost
38
          DB USER: mdb
39
          DB PASSWORD: mdb
          DB NAME: mdb
40
41
           DB PORT: 5432
42
           DB DIALECT: postgres
43
           PGPASSWORD: mdb
44
        run: psql -d postgresql://mdb@localhost/mdb -f install.sql
45
46
       - name: UnitTest
                                  -- Запускаю Unit тесты
47
           go test -coverprofile=coverage1.out -coverpkg=./... -cover ./... &&
49 cat coverage1.out | grep -v mock | grep -v pb.go > cover.out &&go tool cover
50 -func=cover.out
51
52
        - name: IntegrationTests. -- Запускаю Интеграционные тесты
         run:
           cd internal/ratings/integration
```

2) Интеграционные тесты. Тестируются CRUD операции при подключении к реальной БД без заглушек.

```
1 package e2e test
 3 import (
          "context"
          "fmt."
          ratingsDBStorage "github.com/go-park-mail-
 7 ru/2021_1_kekEnd/internal/ratings/repository/dbstorage"
          ratingsUseCase "github.com/go-park-mail-
9 ru/2021 1 kekEnd/internal/ratings/usecase"
          "github.com/jackc/pgx/v4/pgxpool"
          "github.com/stretchr/testify/assert"
11
          "log"
12
13
          "os"
          "testing"
14
15)
16
17
18 func TestRating(t *testing.T) {
         os.Setenv("DB_CONNECT", "postgres://mdb:mdb@127.0.0.1:5432/mdb")
19
20
21
          connStr, connected := os.LookupEnv("DB CONNECT")
22
          if !connected {
23
                  fmt.Println(os.Getwd())
24
                  log.Fatal("Failed to read DB connection data")
25
26
          dbpool, err := pgxpool.Connect(context.Background(), connStr)
27
          assert.NoError(t, err)
28
29
          t.Run("CreateRating and GetRating", func(t *testing.T) {
                  _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
30
31 mdb.movie rating")
32
                  assert.NoError(t, err)
33
34
                  ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
3.5
                  ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
36
37
                  err = ratingsUC.CreateRating("user1", "2", 4)
38
                  assert.NoError(t, err)
39
40
                 rating, err := ratingsUC.GetRating("user1", "2")
41
                  assert.NoError(t, err)
42
43
                 assert.Equal(t, rating.UserID, "user1")
44
                 assert.Equal(t, rating.MovieID, "2")
45
                 assert.Equal(t, rating.Score, 4)
46
         })
47
```

```
48
           t.Run("CreateRating and GetRating", func(t *testing.T) {
                   , err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
 49
 50 mdb.movie rating")
                   assert.NoError(t, err)
 52
 53
                   ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
                  ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
 54
 55
                   err = ratingsUC.CreateRating("user1", "2", 4)
 56
 57
                   assert.NoError(t, err)
 58
 59
                  ratingsRepo2 :=
 60 ratingsDBStorage.NewRatingsRepository(dbpool)
                  ratingsUC2 := ratingsUseCase.NewRatingsUseCase(ratingsRepo2)
 61
 62
 63
                   err = ratingsUC2.CreateRating("user1", "2", 4)
 64
                   assert.Error(t, err)
 65
 66
          })
 67
           t.Run("UpdateRating", func(t *testing.T) {
 68
                   _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
 69
 70 mdb.movie rating")
                   assert.NoError(t, err)
 72
 73
                   ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
 74
                   ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
 7.5
 76
                  err = ratingsUC.CreateRating("user1", "2", 4)
 77
                  assert.NoError(t, err)
 78
 79
                   rating, err := ratingsUC.GetRating("user1", "2")
                  assert.NoError(t, err)
 80
 81
 82
                  assert.Equal(t, rating.UserID, "user1")
                  assert.Equal(t, rating.MovieID, "2")
 83
 84
                   assert.Equal(t, rating.Score, 4)
 85
 86
                  err = ratingsUC.UpdateRating("user1", "2", 8)
 87
                  assert.NoError(t, err)
 88
 89
                  ratingAfterUpdate, err := ratingsUC.GetRating("user1", "2")
 90
                   assert.NoError(t, err)
 91
                   assert.Equal(t, ratingAfterUpdate.UserID, "user1")
 92
 93
                   assert.Equal(t, ratingAfterUpdate.MovieID, "2")
 94
                   assert.Equal(t, ratingAfterUpdate.Score, 8)
 95
          })
 96
           t.Run("DeleteRating", func(t *testing.T) {
 97
                    , err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
 99 mdb.movie rating")
100
                   assert.NoError(t, err)
101
102
                   ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
103
                  ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
104
105
                   err = ratingsUC.CreateRating("user1", "2", 4)
```

```
106
                   assert.NoError(t, err)
107
108
                   rating, err := ratingsUC.GetRating("user1", "2")
109
                   assert.NoError(t, err)
110
                   assert.Equal(t, rating.UserID, "user1")
111
                   assert.Equal(t, rating.MovieID, "2")
112
113
                   assert.Equal(t, rating.Score, 4)
114
115
                   err = ratingsUC.DeleteRating("user1", "2")
                   assert.NoError(t, err)
                    , err = ratingsUC.GetRating("user1", "2")
                   assert.Error(t, err)
           })
    }
```

3) Е2Е тестирование

Тестируются CRUD операции для работы с рейтингом. Используются реальные запросы к серверу.

Запускается сервер, и дальше на него отправляются НТТР-запросы.

```
1 package e2e test
 3 import (
          "context"
 5
          "fmt"
          "github.com/go-park-mail-ru/2021 1 kekEnd/internal/server"
 6
 7
          constants "github.com/go-park-mail-ru/2021 1 kekEnd/pkg/const"
 8
          "github.com/jackc/pgx/v4/pgxpool"
          "github.com/labstack/echo"
 9
          "github.com/stretchr/testify/assert"
10
11
          "io/ioutil"
          "log"
12
13
          "net/http"
          "os"
14
          "strings"
15
16
          "testing"
17)
18
19 func TestE2ERating(t *testing.T) {
          os.Setenv("DB CONNECT", "postgres://mdb:mdb@127.0.0.1:5432/mdb")
20
21
22
          connStr, connected := os.LookupEnv("DB CONNECT")
23
          if !connected {
24
                 fmt.Println(os.Getwd())
25
                 log.Fatal("Failed to read DB connection data")
26
27
          dbpool, err := pqxpool.Connect(context.Background(), connStr)
28
         assert.NoError(t, err)
29
30
31
         app := server.NewApp()
32
33
          go func() {
```

```
34
                  err := app.Run(constants.Port)
35
                  assert.NoError(t, err)
36
          } ()
37
38
          t.Run("CreateRating", func(t *testing.T) {
                  , err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
39
40 mdb.movie rating")
41
                  assert.NoError(t, err)
42
43
44
                  // AddRating
                  reqStrAddRating := `{"movie id":"1", "score": "4"}`
45
                  reqAddRating, err := http.NewRequest(echo.POST,
46
47 fmt.Sprintf("http://localhost:%s/api/v1/ratings", constants.Port),
48 strings.NewReader(regStrAddRating))
49
                  assert.NoError(t, err)
50
                  regAddRating.Header.Set (echo.HeaderContentType,
52 echo.MIMEApplicationJSON)
53
54
                  client := http.Client{}
55
                  response, err := client.Do(reqAddRating)
56
                  assert.NoError(t, err)
57
                  assert.Equal(t, http.StatusCreated, response.StatusCode)
58
59
60
                  // GetRating
61
                  reqGetRating, err := http.NewRequest(echo.GET,
62 fmt.Sprintf("http://localhost:%s/api/v1/ratings/%d", constants.Port, 1),
63 strings.NewReader(""))
64
                  assert.NoError(t, err)
65
66
                  regGetRating.Header.Set(echo.HeaderContentType,
67 echo.MIMEApplicationJSON)
69
                  clientGetRating := http.Client{}
70
                  responseGetRating, err := clientGetRating.Do(reqGetRating)
71
                  assert.NoError(t, err)
72
                  assert.Equal(t, http.StatusOK, responseGetRating.StatusCode)
73
74
                  byteBodyGetRating, err :=
75 ioutil.ReadAll(responseGetRating.Body)
76
                  assert.NoError(t, err)
77
                  assert.Equal(t,
79 \ "username": "user1", "movie id": "1", "score": 4} \ ,
80 strings.Trim(string(byteBodyGetRating), "\n"))
81
                  response.Body.Close()
82
83
84
                  // UpdateRating
85
                  regStrUpdateRating := `{"movie id":"1", "score": "8"}`
                  reqUpdateRating, err := http.NewRequest(http.MethodPut,
86
87 fmt.Sprintf("http://localhost:%s/api/v1/ratings", constants.Port),
88 strings.NewReader(reqStrUpdateRating))
89
                  assert.NoError(t, err)
90
91
```

```
92
                   regUpdateRating.Header.Set(echo.HeaderContentType,
 93 echo.MIMEApplicationJSON)
 94
 95
                   clientUpdateRating := http.Client{}
 96
                   responseUpdateRating, err :=
 97 clientUpdateRating.Do(reqUpdateRating)
                   assert.NoError(t, err)
 99
                   assert.Equal(t, http.StatusOK,
100 responseUpdateRating.StatusCode)
101
102
103
                   // GetRatingAfterUpdate
104
                   clientGetRatingAfterUpdate := http.Client{}
105
                   responseGetRatingAfterUpdate, err :=
106 clientGetRatingAfterUpdate.Do(regGetRating)
107
                  assert.NoError(t, err)
108
                   assert.Equal(t, http.StatusOK,
109 responseGetRatingAfterUpdate.StatusCode)
110
                   byteBodyGetRatingAfterUpdate, err :=
111
112 ioutil.ReadAll(responseGetRatingAfterUpdate.Body)
113
                   assert.NoError(t, err)
114
115
                   assert.Equal(t,
116 `{"username":"user1", "movie id":"1", "score":8}`,
117 strings.Trim(string(byteBodyGetRatingAfterUpdate), "\n"))
118
                   response.Body.Close()
119
120
                   // DeleteRating
121
122
                   reqDeleteRating, err := http.NewRequest(http.MethodDelete,
123 fmt.Sprintf("http://localhost:%s/api/v1/ratings/%d", constants.Port, 1),
124 strings.NewReader(""))
                   assert.NoError(t, err)
                   reqDeleteRating.Header.Set(echo.HeaderContentType,
   echo.MIMEApplicationJSON)
                   clientDeleteRating := http.Client{}
                   responseDeleteRating, err :=
   clientDeleteRating.Do(reqDeleteRating)
                   assert.NoError(t, err)
                   assert.Equal(t, http.StatusOK,
   responseDeleteRating.StatusCode)
                   // GetRatingAfterDelete
                   clientGetRatingAfterDelete := http.Client{}
                   responseGetRatingAfterDelete, err :=
   clientGetRatingAfterDelete.Do(regGetRating)
                   assert.NoError(t, err)
                   assert.Equal(t, http.StatusNotFound,
   responseGetRatingAfterDelete.StatusCode)
                   byteBodyGetRatingAfterDelete, err :=
   ioutil.ReadAll(responseGetRatingAfterDelete.Body)
                   assert.NoError(t, err)
```