



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.

*Москва, 2021 г.*

# Результаты выполнения ЛР

## 1) Запуск CI/CD в GitHub Actions

```
1 name: Simple Build
2 on:
3   push:
4     branches: [master]
5   pull_request:
6     branches: [master]
7
8 jobs:
9   tests:
10    runs-on: ubuntu-latest
11    services:                                -- Поднимаю сервер Постгреса
12      postgres:
13        image: postgres:12
14        env:
15          POSTGRES_USER: mdb
16          POSTGRES_PASSWORD: mdb
17          POSTGRES_DB: mdb
18        ports:
19          - 5432:5432
20        options: >-
21          --health-cmd pg_isready
22          --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
33        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
40          DB_NAME: mdb
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        run: |
48          go test -coverprofile=coveragel.out -coverpkg=./... -cover ./... &&
49          cat coveragel.out | grep -v _mock | grep -v pb.go > cover.out &&go tool cover
50          -func=cover.out
51
52      - name: IntegrationTests.             -- Запускаю Интеграционные тесты
53        run: |
54          cd internal/ratings/integration
```

```

55         go test
56
57     - name: E2ETests                -- Запускаю E2E тесты
58     run: |
59         cd internal/ratings/e2e
60         go test

```

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

```

1 package e2e_test
2
3 import (
4     "context"
5     "fmt"
6     ratingsDBStorage "github.com/go-park-mail-
7 ru/2021_1_kekEnd/internal/ratings/repository/dbstorage"
8     ratingsUseCase "github.com/go-park-mail-
9 ru/2021_1_kekEnd/internal/ratings/usecase"
10    "github.com/jackc/pgx/v4/pgxpool"
11    "github.com/stretchr/testify/assert"
12    "log"
13    "os"
14    "testing"
15 )
16
17
18 func TestRating(t *testing.T) {
19     os.Setenv("DB_CONNECT", "postgres://mdb:mdb@127.0.0.1:5432/mdb")
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) {
30         _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
31 mdb.movie_rating")
32         assert.NoError(t, err)
33
34         ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
35         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) {
49             _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
50 mdb.movie_rating")
51             assert.NoError(t, err)
52
53             ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
54             ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
55
56             err = ratingsUC.CreateRating("user1", "2", 4)
57             assert.NoError(t, err)
58
59             ratingsRepo2 :=
60 ratingsDBStorage.NewRatingsRepository(dbpool)
61             ratingsUC2 := ratingsUseCase.NewRatingsUseCase(ratingsRepo2)
62
63             err = ratingsUC2.CreateRating("user1", "2", 4)
64             assert.Error(t, err)
65
66         })
67
68         t.Run("UpdateRating", func(t *testing.T) {
69             _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
70 mdb.movie_rating")
71             assert.NoError(t, err)
72
73             ratingsRepo := ratingsDBStorage.NewRatingsRepository(dbpool)
74             ratingsUC := ratingsUseCase.NewRatingsUseCase(ratingsRepo)
75
76             err = ratingsUC.CreateRating("user1", "2", 4)
77             assert.NoError(t, err)
78
79             rating, err := ratingsUC.GetRating("user1", "2")
80             assert.NoError(t, err)
81
82             assert.Equal(t, rating.UserID, "user1")
83             assert.Equal(t, rating.MovieID, "2")
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
92             assert.Equal(t, ratingAfterUpdate.UserID, "user1")
93             assert.Equal(t, ratingAfterUpdate.MovieID, "2")
94             assert.Equal(t, ratingAfterUpdate.Score, 8)
95         })
96
97         t.Run("DeleteRating", func(t *testing.T) {
98             _, 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
111         assert.Equal(t, rating.UserID, "user1")
112         assert.Equal(t, rating.MovieID, "2")
113         assert.Equal(t, rating.Score, 4)
114
115         err = ratingsUC.DeleteRating("user1", "2")
116         assert.NoError(t, err)
117
118         _, err = ratingsUC.GetRating("user1", "2")
119         assert.Error(t, err)
120     })
121 }

```

### 3) E2E тестирование

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

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

```

1 package e2e_test
2
3 import (
4     "context"
5     "fmt"
6     "github.com/go-park-mail-ru/2021_1_kekEnd/internal/server"
7     constants "github.com/go-park-mail-ru/2021_1_kekEnd/pkg/const"
8     "github.com/jackc/pgx/v4/pgxpool"
9     "github.com/labstack/echo"
10    "github.com/stretchr/testify/assert"
11    "io/ioutil"
12    "log"
13    "net/http"
14    "os"
15    "strings"
16    "testing"
17 )
18
19 func TestE2ERating(t *testing.T) {
20     os.Setenv("DB_CONNECT", "postgres://mdb:mdb@127.0.0.1:5432/mdb")
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 := pgxpool.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) {
39         _, err = dbpool.Exec(context.Background(), "TRUNCATE TABLE
40 mdb.movie_rating")
41         assert.NoError(t, err)
42
43
44         // AddRating
45         reqStrAddRating := `{"movie_id": "1", "score": "4"}`
46         reqAddRating, err := http.NewRequest(echo.POST,
47 fmt.Sprintf("http://localhost:%s/api/v1/ratings", constants.Port),
48 strings.NewReader(reqStrAddRating))
49         assert.NoError(t, err)
50
51         reqAddRating.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         reqGetRating.Header.Set(echo.HeaderContentType,
67 echo.MIMEApplicationJSON)
68
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
78         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         reqStrUpdateRating := `{"movie_id": "1", "score": "8"}`
86         reqUpdateRating, err := http.NewRequest(http.MethodPut,
87 fmt.Sprintf("http://localhost:%s/api/v1/ratings", constants.Port),
88 strings.NewReader(reqStrUpdateRating))
89         assert.NoError(t, err)
90
91

```

```

92         reqUpdateRating.Header.Set(echo.HeaderContentType,
93 echo.MIMEApplicationJSON)
94
95         clientUpdateRating := http.Client{}
96         responseUpdateRating, err :=
97 clientUpdateRating.Do(reqUpdateRating)
98         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(reqGetRating)
107         assert.NoError(t, err)
108         assert.Equal(t, http.StatusOK,
109 responseGetRatingAfterUpdate.StatusCode)
110
111         byteBodyGetRatingAfterUpdate, err :=
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
121         // DeleteRating
122         reqDeleteRating, err := http.NewRequest(http.MethodDelete,
123 fmt.Sprintf("http://localhost:%s/api/v1/ratings/%d", constants.Port, 1),
124 strings.NewReader(""))
125         assert.NoError(t, err)
126
127         reqDeleteRating.Header.Set(echo.HeaderContentType,
128 echo.MIMEApplicationJSON)
129
130         clientDeleteRating := http.Client{}
131         responseDeleteRating, err :=
132 clientDeleteRating.Do(reqDeleteRating)
133         assert.NoError(t, err)
134         assert.Equal(t, http.StatusOK,
135 responseDeleteRating.StatusCode)
136
137
138         // GetRatingAfterDelete
139         clientGetRatingAfterDelete := http.Client{}
140         responseGetRatingAfterDelete, err :=
141 clientGetRatingAfterDelete.Do(reqGetRating)
142         assert.NoError(t, err)
143         assert.Equal(t, http.StatusNotFound,
144 responseGetRatingAfterDelete.StatusCode)
145
146         byteBodyGetRatingAfterDelete, err :=
147 ioutil.ReadAll(responseGetRatingAfterDelete.Body)
148         assert.NoError(t, err)

```

```
        assert.Equal(t, "",
strings.Trim(string(byteBodyGetRatingAfterDelete), "\n"))
        response.Body.Close()
    })
}
```