

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ (МАСТЕР) РАД

**Упоредни преглед карактеристика
монолитних и микросрвисних апликација у
Јава ЕЕ окружењу**

Ментор:
др Милош Милић

Студент:
Денис Кужнер 2019/3713

Београд, 2021. године

Попис слика

Слика 1. Јава технологија [5].....	4
Слика 2. Позадина извршавања Јава програма [7] [8].....	5
Слика 3. Однос између JVM, JRE и JDK [9]	5
Слика 4. Spring модули [17]	12
Слика 5. Spring Cloud Config архитектура.....	16
Слика 6. Spring Cloud архитектура	18
Слика 7. Spring Security - Basic аутентификација [25]	19
Слика 8. Spring Security - аутентификација путем форме [25].....	20
Слика 9. Spring Security - JWT аутентификација [25]	20
Слика 10. Детаљан приказ Hibernate архитектуре [27]	22
Слика 11. Структура Angular компоненте [32].....	27
Слика 12. Elastic Stack архитектура [34]	31
Слика 13. Elastic Stack архитектура у сложеним системима [34]	31
Слика 14. Монолитна архитектура	33
Слика 15. Модуларни монолит	34
Слика 16. Скалирање монолитне апликације.....	35
Слика 17. Микросервисна архитектура	37
Слика 18. API Gateway	38
Слика 19. Скалирање микросрвиса.....	40
Слика 20. Комплексност микросрвисних система [41]	42
Слика 21. Поступна миграција функционалности из монолита у микросрвис	44
Слика 22. Организационе структуре монолитне и микросрвисне архитектуре	45
Слика 23. Циљ декомпозиције монолитног система	45
Слика 24. Дељење података између монолита и микросрвиса.....	46
Слика 25. Проблем дељене базе података између микросрвиса са предлогом решења.....	46
Слика 26. Спајање података у монолитној и микросрвисној архитектури	47
Слика 27. Фазе развоја софтверског система по Лармановој методи [4]	49
Слика 28. Модел случајева коришћења.....	51
Слика 29. Логичка структура и понашање софтверског система	95
Слика 30. Тронивојска архитектура [4].....	96
Слика 31. Структура корисничког интерфејса [4]	96
Слика 32. Форма за регистрацију	98
Слика 33. Исправно попуњена форма за регистрацију.....	99
Слика 34. Порука о успешној регистрацији.....	99
Слика 35. Порука о неуспешној регистрацији.....	99
Слика 36. Форма за пријављивање.....	100
Слика 37. Исправно попуњена форма за пријављивање	100
Слика 38. Порука о успешном пријављивању на систем	101
Слика 39. Порука о неуспешном пријављивању на систем	101
Слика 40. Форма за рад са мечевима	102
Слика 41. Преглед меча.....	103
Слика 42. Порука о неуспешном прегледу меча	103
Слика 43. Форма за креирање тима.....	104
Слика 44. Исправно унети подаци о тиму	105
Слика 45. Порука о успешном креирању тима	106
Слика 46. Порука о неуспешном креирању тима	106
Слика 47. Форма за рад са лигама.....	107
Слика 48. Преглед тима.....	108

Слика 49. Порука о неуспешном прегледу тима	108
Слика 50. Форма за рад са тимом.....	109
Слика 51. Измена тима	110
Слика 52. Порука о неуспешној измени тима	110
Слика 53. Исправно унети подаци о лиги	111
Слика 54. Порука о успешном креирању лиге	111
Слика 55. Порука о неуспешном креирању лиге	111
Слика 56. Исправно унети подаци за приступање лиги.....	112
Слика 57. Порука о успешном приступању лиги.....	112
Слика 58. Порука о неуспешном приступању лиги.....	112
Слика 59. Брисање лиге	113
Слика 60. Порука о успешном брисању лиге.....	113
Слика 61. Порука о неуспешном брисању лиге.....	113
Слика 62. Форма за рад са играчима	114
Слика 63. Преглед играча.....	115
Слика 64. Порука о неуспешном учитавању играча	115
Слика 65. Измена играча	116
Слика 66. Порука о успешној измени играча	117
Слика 67. Порука о неуспешној измени играча	117
Слика 68. Порука о успешном брисању играча	118
Слика 69. Порука о неуспешном брисању играча	118
Слика 70. Форма за рад са клубовима.....	119
Слика 71. Преглед клуба	120
Слика 72. Порука о неуспешном прегледу клуба	120
Слика 73. Порука о успешном брисању клуба.....	121
Слика 74. Порука о неуспешном брисању клуба.....	121
Слика 75. Коначан изглед архитектуре софтверског система.....	136
Слика 76. Компоненте клијентске апликације.....	137
Слика 77. Модели клијентске апликације	138
Слика 78. Сервиси клијентске апликације	138
Слика 79. Пакет domain	139
Слика 80. Пакет repository	139
Слика 81. Пакет service.....	139
Слика 82. Пакет rest	139
Слика 83. Unit тестови	140
Слика 84. Архитектура монолитне апликације	143
Слика 85. Коначан изглед архитектуре микросрвиса football-fantasy-user-service.....	144
Слика 86. Архитектура микросрвисне апликације	145
Слика 87. JMeter тест план	146
Слика 88. Време извршења - монолитна (JVM) и микросрвисна (JVM) апликација ..	150
Слика 89. Кашњење - монолитна (JVM) и микросрвисна (JVM) апликација	151
Слика 90. Проценат успешности - монолитна (JVM) апликација.....	151
Слика 91. Проценат успешности - микросрвисна (JVM) апликација	152
Слика 92. Време изградње извршног програма - монолитна (JVM) и микросрвисина (JVM) апликација	152
Слика 93. Време покретања апликације - монолитна (JVM) и микросрвисина (JVM) апликација	153
Слика 94. Употреба heap меморије - монолитна (JVM) и микросрвисина (JVM) апликација	153
Слика 95. Употреба процесора - монолитна (JVM) апликација	154
Слика 96. Употреба процесора - микросрвисна (JVM) апликација.....	154

Слика 97. Време извршења - монолитна (GraalVM) и микросервисна (GraalVM) апликација	155
Слика 98. Кашњење - монолитна (GraalVM) и микросервисна (GraalVM) апликација	156
Слика 99. Проценат успешности - монолитна (GraalVM) апликација.....	156
Слика 100. Проценат успешности - микросервисна (GraalVM) апликација.....	157
Слика 101. Време изградње извршног програма - монолитна (GraalVM) и микросервисна (GraalVM) апликација	157
Слика 102. Време покретања апликације - монолитна (GraalVM) и микросервисна (GraalVM) апликација	158
Слика 103. Употреба heap меморије - монолитна (GraalVM) и микросервисна (GraalVM) апликација	158
Слика 104. Употреба процесора - монолитна (GraalVM) апликација	159
Слика 105. Употреба процесора - микросервисна (GraalVM) апликација	159
Слика 106. Време извршења - монолитна (JVM) и монолитна (GraalVM) апликација	160
Слика 107. Кашњење - монолитна (JVM) и монолитна (GraalVM) апликација.....	161
Слика 108. Време изградње извршног програма - монолитна (JVM) и монолитна (GraalVM) апликација	162
Слика 109. Време покретања апликације - монолитна (JVM) и монолитна (GraalVM) апликација	162
Слика 110. Употреба heap меморије - монолитна (JVM) и монолитна (GraalVM) апликација	163
Слика 111. Време извршења - микросервисна (JVM) и микросервисна (GraalVM) апликација	164
Слика 112. Кашњење - микросервисна (JVM) и микросервисна (GraalVM) апликација	165
Слика 113. Време изградње извршног програма - микросервисна (JVM) и микросервисна (GraalVM) апликација	166
Слика 114. Време покретања апликације - микросервисна (JVM) и микросервисна (GraalVM) апликација	166
Слика 115. Употреба heap меморије - микросервисна (JVM) и микросервисна (GraalVM) апликација	167
Слика 116. Време извршења – све апликације	168
Слика 117. Кашњење - све апликације	169
Слика 118. Проценат успешности - све апликације	170
Слика 119. Време изградње извршног програма – све апликације	171
Слика 120. Време покретања апликације – све апликације	172
Слика 121. Употреба heap меморије – све апликације	173
Слика 122. Употреба процесора – све апликације	174

Попис дијаграма

Дијаграм 1. Дијаграм секвенци СК - Регистрација корисника	67
Дијаграм 2. Дијаграм секвенци СК - Регистрација корисника, алтернативни сценарио 2.1.....	67
Дијаграм 3. Дијаграм секвенци СК - Пријављивање на систем	68
Дијаграм 4. Дијаграм секвенци СК - Пријављивање на систем, алтернативни сценарио 2.1.....	68
Дијаграм 5. Дијаграм секвенци СК - Преглед меча	69
Дијаграм 6. Дијаграм секвенци СК - Преглед меча, алтернативни сценарио 4.1.	69
Дијаграм 7. Дијаграм секвенци СК - Креирање тима (Сложен СК)	70
Дијаграм 8. Дијаграм секвенци СК - Креирање тима (Сложен СК), алтернативни сценарио 4.1.....	70
Дијаграм 9. Дијаграм секвенци СК - Преглед тима.....	71
Дијаграм 10. Дијаграм секвенци СК - Преглед тима, алтернативни сценарио 4.1.	71
Дијаграм 11. Дијаграм секвенци СК - Измена тима.....	72
Дијаграм 12. Дијаграм секвенци СК - Измена тима, алтернативни сценарио 6.1.....	72
Дијаграм 13. Дијаграм секвенци СК - Креирање лиге	74
Дијаграм 14. Дијаграм секвенци СК - Креирање лиге, алтернативни сценарио 2.1.	74
Дијаграм 15. Дијаграм секвенци СК - Креирање чланства лиге	75
Дијаграм 16. Дијаграм секвенци СК - Креирање чланства лиге, алтернативни сценарио 2.1.....	75
Дијаграм 17. Дијаграм секвенци СК - Брисање лиге	76
Дијаграм 18. Дијаграм секвенци СК - Брисање лиге, алтернативни сценарио 4.1.	76
Дијаграм 19. Дијаграм секвенци СК - Преглед играча.....	77
Дијаграм 20. Дијаграм секвенци СК - Преглед играча, алтернативни сценарио 4.1.	77
Дијаграм 21. Дијаграм секвенци СК - Измена играча	78
Дијаграм 22. Дијаграм секвенци СК - Измена играча, алтернативни сценарио 4.1.....	79
Дијаграм 23. Дијаграм секвенци СК - Измена играча, алтернативни сценарио 6.1.....	79
Дијаграм 24. Дијаграм секвенци СК - Брисање играча.....	80
Дијаграм 25. Дијаграм секвенци СК - Брисање играча, алтернативни сценарио 4.1. ...	81
Дијаграм 26. Дијаграм секвенци СК - Брисање играча, алтернативни сценарио 6.1. ...	81
Дијаграм 27. Дијаграм секвенци СК - Преглед клуба	82
Дијаграм 28. Дијаграм секвенци СК - Преглед клуба, алтернативни сценарио 4.1.	82
Дијаграм 29. Дијаграм секвенци СК - Брисање клуба	83
Дијаграм 30. Дијаграм секвенци СК - Брисање клуба, алтернативни сценарио 4.1.	83
Дијаграм 31. Концептуални модел	88
Дијаграм 32. Уговор УГ1: GetClubs	123
Дијаграм 33. Уговор УГ2: SaveUser	123
Дијаграм 34. Уговор УГ3: Login	123
Дијаграм 35. Уговор УГ4: GetMatches	124
Дијаграм 36. Уговор УГ5: GetMatch	124
Дијаграм 37. Уговор УГ6: GetPlayers	124
Дијаграм 38. Уговор УГ7: SaveTeam	125
Дијаграм 39. Уговор УГ8: GetLeagues	125
Дијаграм 40. Уговор УГ9: GetTeam	125
Дијаграм 41. Уговор УГ10: UpdateTeam	126
Дијаграм 42. Уговор УГ11: SaveLeague	126
Дијаграм 43. Уговор УГ12: SaveTeamLeagueMembership	126
Дијаграм 44. Уговор УГ13: DeleteLeague	127
Дијаграм 45. Уговор УГ14: GetPlayer	127

Дијаграм 46. Уговор УГ15: UpdatePlayer	127
Дијаграм 47. Уговор УГ16: DeletePlayer	128
Дијаграм 48. Уговор УГ17: GetClub.....	128
Дијаграм 49. Уговор УГ18: DeleteClub	128

Попис табела

Табела 1. Табела User	90
Табела 2. Табела Role	90
Табела 3. Табела UserRole.....	90
Табела 4. Табела Club	90
Табела 5. Табела Player	91
Табела 6. Табела Team	91
Табела 7. Табела TeamPlayer	91
Табела 8. Табела League.....	92
Табела 9. Табела TeamLeagueMembership	92
Табела 10. Табела Gameweek.....	92
Табела 11. Табела TeamGameweekPerformance	92
Табела 12. PlayerGameweekPerformance	93
Табела 13. Табела Match.....	93
Табела 14. Табела MatchEvent.....	93
Табела 15. Табела Goal.....	93
Табела 16. Табела Card.....	94
Табела 17. Табела MinutesPlayed	94
Табела 18. Табела Substitution.....	94
Табела 19. Складиште података: Табела User	131
Табела 20. Складиште података: Табела Role.....	131
Табела 21. Складиште података: Табела UserRole	131
Табела 22. Складиште података: Табела Club.....	131
Табела 23. Складиште података: Табела Player	132
Табела 24. Складиште података: Табела Team	132
Табела 25. Складиште података: Табела TeamPlayer	132
Табела 26. Складиште података: Табела League	133
Табела 27. Складиште података: Табела TeamLeagueMembership	133
Табела 28. Складиште података: Табела Gameweek	133
Табела 29. Складиште података: Табела TeamGameweekPerformance	133
Табела 30. Складиште података: Табела PlayerGameweekPerformance	133
Табела 31. Складиште података: Табела Match	134
Табела 32. Складиште података: Табела MatchEvent	134
Табела 33. Складиште података: Табела Goal	134
Табела 34. Складиште података: Табела Card	134
Табела 35. Складиште података: Табела MinutesPlayed.....	135
Табела 36. Складиште података: Табела Substitution	135

Садржај

1. Увод	1
2. Преглед коришћених технологија.....	4
2.1. Јава технологија	4
2.1.1. Јава платформа	4
2.1.2. Програмски језик Јава	6
2.2. GraalVM	8
2.2.1. Native image	9
2.3. Spring	12
2.4. Spring Boot.....	14
2.5. Spring Cloud.....	15
2.5.1. Spring Cloud Config	15
2.5.2. Spring Cloud Netflix	16
2.5.3. Spring Cloud Gateway	17
2.6. Spring Security	19
2.7. Hibernate.....	22
2.8. Spring Data JPA	24
2.9. JUnit.....	26
2.10. Angular.....	27
2.10.1. Angular-Cli	28
2.10.2. Angular Material	29
2.10.3. TypeScript.....	29
2.11. Elastic Stack.....	30
3. Преглед карактеристика монолитне и микросервисне архитектуре.....	32
3.1. Монолитна архитектура.....	33
3.2. Микросервисна архитектура	37
3.3. Миграција монолитне у микросервисну архитектуру	43
4. Студијски пример	49
4.1. Прикупљање захтева од корисника	50
4.1.1. Вербални опис модела	50
4.1.2. Случајеви коришћења.....	50
СК1: Случај коришћења – Регистрација корисника.....	52
СК2: Случај коришћења - Пријављивање на систем	53
СК3: Случај коришћења – Преглед меча	54
СК4: Случај коришћења – Креирање тима (Сложен СК)	55
СК5: Случај коришћења – Преглед тима	56
СК6: Случај коришћења – Измена тима	57
СК7: Случај коришћења – Креирање лиге.....	58

СК8: Случај коришћења – Креирање чланства лиге	59
СК9: Случај коришћења – Брисање лиге	60
СК10: Случај коришћења – Преглед играча	61
СК11: Случај коришћења – Измена играча	62
СК12: Случај коришћења – Брисање играча	63
СК13: Случај коришћења – Преглед клуба.....	64
СК14: Случај коришћења – Брисање клуба	65
4.2. Анализа	66
4.2.1. Понашање софтверског система – Системски дијаграм секвенци	66
ДС1: Дијаграм секвенци случаја коришћења – Регистрација корисника.....	67
ДС2: Дијаграм секвенци случаја коришћења - Пријављивање на систем	68
ДС3: Дијаграм секвенци случаја коришћења – Преглед меча	69
ДС4: Дијаграм секвенци случаја коришћења – Креирање тима (Сложен СК)	70
ДС5: Дијаграм секвенци случаја коришћења – Преглед тима	71
ДС6: Дијаграм секвенци случаја коришћења – Измена тима	72
ДС7: Дијаграм секвенци случаја коришћења – Креирање лиге.....	74
ДС8: Дијаграм секвенци случаја коришћења – Креирање чланства лиге.....	75
ДС9: Дијаграм секвенци случаја коришћења – Брисање лиге	76
ДС10: Дијаграм секвенци случаја коришћења – Преглед играча	77
ДС11: Дијаграм секвенци случаја коришћења – Измена играча	78
ДС12: Дијаграм секвенци случаја коришћења – Брисање играча	80
ДС13: Дијаграм секвенци случаја коришћења – Преглед клуба.....	82
ДС14: Дијаграм секвенци случаја коришћења – Брисање клуба	83
4.2.2. Понашање софтверског система – Дефинисање уговора о системским операцијама.....	85
Уговор УГ1: GetClubs	85
Уговор УГ2: SaveUser	85
Уговор УГ3: Login	85
Уговор УГ4: GetMatches	85
Уговор УГ5: GetMatch.....	85
Уговор УГ6: GetPlayers.....	85
Уговор УГ7: SaveTeam.....	86
Уговор УГ8: GetLeagues	86
Уговор УГ9: GetTeam	86
Уговор УГ10: UpdateTeam	86
Уговор УГ11: SaveLeague	86

Уговор УГ12: SaveTeamLeagueMembership	86
Уговор УГ13: DeleteLeague	86
Уговор УГ14: GetPlayer	86
Уговор УГ15: UpdatePlayer	87
Уговор УГ16: DeletePlayer	87
Уговор УГ17: GetClub	87
Уговор УГ18: DeleteClub	87
4.2.3. Структура софтверског система – Концептуални (доменски) модел	88
4.2.4 Структура софтверског система – Релациони модел	89
4.3. Пројектовање	96
4.3.1. Пројектовање корисничког интерфејса	96
СК1: Случај коришћења – Регистрација корисника	98
СК2: Случај коришћења - Пријављивање на систем	100
СК3: Случај коришћења – Преглед меча	102
СК4: Случај коришћења – Креирање тима (Сложен СК)	104
СК5: Случај коришћења – Преглед тима	107
СК6: Случај коришћења – Измена тима	109
СК7: Случај коришћења – Креирање лиге	111
СК8: Случај коришћења – Креирање чланства лиге	112
СК9: Случај коришћења – Брисање лиге	113
СК10: Случај коришћења – Преглед играча	114
СК11: Случај коришћења – Измена играча	116
СК12: Случај коришћења – Брисање играча	118
СК13: Случај коришћења – Преглед клуба	119
СК14: Случај коришћења – Брисање клуба	121
4.3.2. Пројектовање апликационе логике	122
4.3.2.1. Контролер апликационе логике	122
4.3.2.2. Пословна логика	123
4.3.3. Пројектовање брокера базе података	130
4.3.4. Пројектовање складишта података	131
4.3.5. Коначан изглед архитектуре софтверског система	136
4.4. Имплементација	137
4.5. Тестирање	140
5. Упоредна анализа примене монолитне и микросервисне архитектуре у развоју софтверских система	141
5.1. Формулација проблема	142
5.2. Предлог решења	143

5.3. Мерење резултата	146
5.4. Резултат анализе.....	150
5.4.1. Поређење монолитне (JVM) и микросервисне (JVM) апликације.....	150
5.4.1.1. Време извршења кључних метода	150
5.4.1.2. Кашњење	151
5.4.1.3. Проценат успешности	151
5.4.1.4. Време изградње извршног програма.....	152
5.4.1.5. Време покретања апликације	153
5.4.1.6. Употреба heap меморије	153
5.4.1.7. Употреба процесора	154
5.4.2. Поређење монолитне (GraalVM) и микросервисне (GraalVM) апликације ..	155
5.4.2.1. Време извршења кључних метода	155
5.4.2.2. Кашњење	156
5.4.2.3. Проценат успешности	156
5.4.2.4. Време изградње извршног програма.....	157
5.4.2.5. Време покретања апликације	158
5.4.2.6. Употреба heap меморије	158
5.4.2.7. Употреба процесора	159
5.4.3. Поређење монолитне (JVM) и монолитне (GraalVM) апликације	160
5.4.3.1. Време извршења кључних метода	160
5.4.3.2. Кашњење	161
5.4.3.3. Проценат успешности	161
5.4.3.4. Време изградње извршног програма.....	161
5.4.3.5. Време покретања апликације	162
5.4.3.6. Употреба heap меморије	163
5.4.3.7. Употреба процесора	163
5.4.4. Поређење микросервисне (JVM) и микросервисне (GraalVM) апликације..	164
5.4.4.1. Време извршења кључних метода	164
5.4.4.2. Кашњење	165
5.4.4.3. Проценат успешности	165
5.4.4.4. Време изградње извршног програма.....	165
5.4.4.5. Време покретања апликације	166
5.4.4.6. Употреба heap меморије	167
5.4.4.7. Употреба процесора	167
5.5. Дискусија	168
5.5.1. Време извршења кључних метода	168

5.5.2. Кашњење	169
5.5.3. Проценат успешности	170
5.5.4. Време изградње извршног програма	171
5.5.5. Време покретања апликације	172
5.5.6. Употреба heap меморије.....	173
5.5.7. Употреба процесора.....	174
5.6. Ограничевања евалуације	175
6. Закључак.....	176
7. Литература	179

1. Увод

Технологија је доживела експоненцијални напредак у претходним деценијама што је довело до стварања нових концепата и правца у развоју софтвера. Константно се проналазе све бољи начини за изградњу софтверских система. Уважавањем знања из прошлости и прихватањем новог таласа технологија настају ИТ системи који носе већу вредност како за програмере тако и за кориснике.

Једна од првих великих одлука које треба донети приликом развоја софтверског система је најчешће везана за његову архитектуру. На највишем нивоу апстракције, архитектура система описује његове кључне компоненте, као и начин на који су оне међусобно повезане.[1] Добро постављена архитектура дугорочно има значајан утицај на успех система у целини, његове перформансе, квалитет и лакоћу одржавања. На тај начин се проактивно смањује вероватноћа појављивања потенцијалних проблема у будућности. Са еволуцијом софтверског инжењерства упоредно је као дисциплина расла и софтверска архитектура. Одговарањем на промене које је ново време доносило мењале су се технике, стилови и правци размишљања везани за архитектуру. Са тим у вези, често се у свету софтверског инжењерства наилази на два кључна термина која описују архитектуру: монолити и микросервиси.

Претходне генерације софтверских компанија су се ослањале искључиво на чисту монолитну архитектуру апликација. Овакве апликације представљају традиционалан избор. Првобитно су то биле јединствене, недељиве јединице које обједињују све функционалности, пословну логику и ресурсе на једном месту. Данас и оне могу имати више модула. Погодне су када постоји потреба за мањом апликацијом за интерну употребу, без сложене пословне логике и флексибилности, или када се ради о раним фазама пројекта. Већина данашњих успешних апликација су започете као монолити. Међутим, поред бројних предности, популарност монолитне архитектуре опада због разних проблема које доноси са собом. Новонастали, модерни захтеви тржишта, жеље клијената и остали фактори захтевали су темељно редизајнирање традиционалних метода монолитних апликација. Сматра се да монолити више нису у стању да адекватно одговоре на брзину тржишног надметања данашњице. Као решење се појавила микросервисна архитектура, која нуди изузетну агилност и пружа могућност брзих измена и надоградњи система које су неопходне како би се одржала конкурентност у пословању. Сем Њуман дефинише микросервисе на следећи начин: „Микросервиси су мали сервиси који раде заједно“.[2] Основна идеја која стоји иза ове архитектуре јесте креирање мањих, индивидуално управљивих јединица које су међусобно повезане. Сваки микросервис је мала апликација која има сопствену пословну логику, базу података и конкретну функцију коју обавља. Популарности микросервисне архитектуре је

посебно допринела чињеница да су неке од водећих светских компанија, као што су Google, Netflix, Spotify, eBay и Amazon, своје монолитне системе замениле микросервисним.[3] Њихов пример следе и многе мање компаније надајући се да ће постићи приближан ниво успеха. Међутим, као и монолитна, микросервисна архитектура има бројне недостатке.

На основу кратког увода, може се приметити да најбољи универзални избор између монолита и микросрвиса не постоји и да остају отворена одређена питања везана за архитектуру софтверских система. Да ли је монолитна архитектура превазиђена? Да ли је вредно трансформисање целог система у микросрвисну архитектуру? Да ли по сваку цену инсистирати на микросрвисима? Са тим у вези, главна проблематика овог рада јесте детаљно представљање карактеристика монолитне и микросрвисне архитектуре, као и њихово међусобно поређење на конкретном примеру, по унапред дефинисаним, мерљивим критеријумима.

Годинама уназад је све више присутна тенденција великих компанија да развијају сопствене имплементације Јава виртуелне машине. Тако данас постоје Alibaba Dragonwell, Amazon Corretto, Oracle GraalVM, као и многе друге које показују изузетно добре резултате. Са тим у вези, у оквиру овог рада је тестирано и понашање монолита и микросрвиса у раду са GraalVM виртуелном машином

Уводни део рада је посвећен детаљном опису технологија које су коришћене за израду софтверских система студијског примера. Како је данас доступан широк избор технологија, веома је битно одабрати оне које ће на најбољи начин задовољити потребе корисника и клијената. Серверске апликације, обе архитектуре, су развијене у Јава ЕЕ окружењу, употребом Spring Boot радног оквира. За потребе развоја микросрвиса је коришћен радни оквир Spring Cloud. Као алат за објектно-релационо мапирање и перзистенцију података коришћен је Spring Data JPA. За сигурност и заштиту апликација је коришћен Spring Security. За тестирање је коришћен JUnit оквир. За развој клијентске апликације је коришћен радни оквир Angular, који се базира на програмском језику TypeScript, уз додатну употребу Angular Material и Bootstrap библиотека. За потребе поређења, мерења и визуелизације постигнутих резултата наведених архитектура коришћен је Elastic Stack, састављен од следећих технологија: Metricbeat, Filebeat, Logstash, Elasticsearch и Kibana. Metricbeat има функцију сакупљања и слања системских метрика. Filebeat и Logstash имају улогу читања, трансформације и слања апликативних логова у Elasticsearch базу података. Kibana је коришћена за презентациони слој и визуелизацију резултата тестирања.

Након тога је дат детаљан теоријски опис карактеристика монолитне и микросрвисне архитектуре. Опширно су представљена њихова својства, са свим предностима и

манама. Детаљно је описан и комплетан поступак миграције монолитних у микросервисне системе.

Затим је представљен развој софтверског система студијског примера применом упрошћене Ларманове методе. Она подразумева да се развој софтверског система састоји од следећих пет фаза: прикупљање захтева од корисника, анализа, пројектовање, имплементација и тестирање.[4] На истом студијском примеру Football Fantasy игре су развијени и монолитни и микросервисни системи. У питању је једна од најпопуларнијих спортских веб апликација данашњице чији се број корисника мери милионима. Она представља адекватан студијски пример јер је довољна сложена за развој применом монолитне, као и микросервисне архитектуре.

Након тога следи упоредна анализа употребе наведених архитектура у развоју софтверских система по унапред дефинисаном скупу критеријума. Прецизно су наведени критеријуми поређења, метрике, методологије и поступци коришћени у истраживању, након чега су представљени добијени резултати заједно са визуелизацијама и донесеним закључцима. За потребе тестирања је, помоћу алата JMeter, извршена симулација различитог броја корисничких захтева ка апликацијама које су развијене у оквиру студијског примера, како би се забележили постигнути резултати.

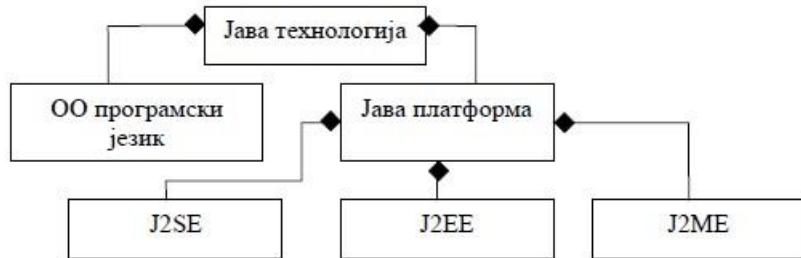
На самом крају је дат изведенни закључак, као и списак референтне литературе коришћене приликом писања рада.

2. Преглед коришћених технологија

У овом одељку су детаљно описане технологије коришћене приликом израде софтверских система студијског примера.

2.1. Јава технологија

Јава је широко распрострањена технологија која не представља само програмски језик, већ је такође и софтверска платформа, што се може видети на слици 1. [5]

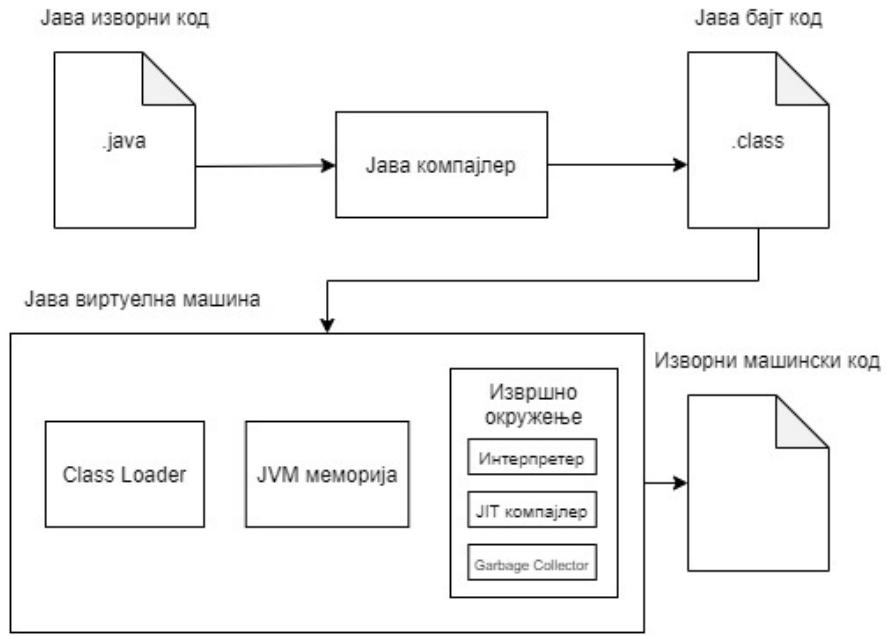


Слика 1. Јава технологија [5]

2.1.1. Јава платформа

Јава платформа је окружење које корисницима омогућава развој и извршавање програма написаних Јава програмским језиком. Она укључује Јава виртуелну машину, Јава API и скуп стандардних библиотека. Јава је платформски независна од било ког хардвера или оперативног система. То значи да се Јава код написан на одређеној платформи (оперативном систему) може извршити и на другим платформама без икаквих измена. Разлог за то јесте чињеница да за сваки оперативни систем постоји посебна Јава виртуелна машина, која је предуслов за извршавање Јава програма.

Позадина извршавања Јава програма је упрошћено приказана на слици 2. Јава компајлер конвертује Јава изворни код у бајт код. Као резултат овог корака се од фајлова са екstenзијом .java добијају .class фајлови. Бајт код још увек није разумљив платформи на којој се покреће, већ је намењен Јава виртуелној машини. Прецизније, он је увек исти без обзира на хардвер или оперативни систем на којима се покреће програм. JVM у следећем кораку интерпретира бајт код у наредбе конкретног оперативног система.[6] Прецизније, битне компоненте које чине JVM су: Class Loader, JVM меморија и извршно окружење (енгл. Execution Engine). Неке од улога које има Class Loader су да учита .class фајлове у меморију, да верификује валидност бајт кода и иницијализује статичка поља. Извршно окружење чине интерпретер, JIT компајлер и garbage collector. Just-In-Time (JIT) компајлер значајно оптимизује процес интерпретирања конвертовањем комплетног бајт кода у изворни машински код. Сваки пут када се поново позове одређена метода, он обезбеђује постојећи машински код тако да поновна интерпретација није потребна.[7]

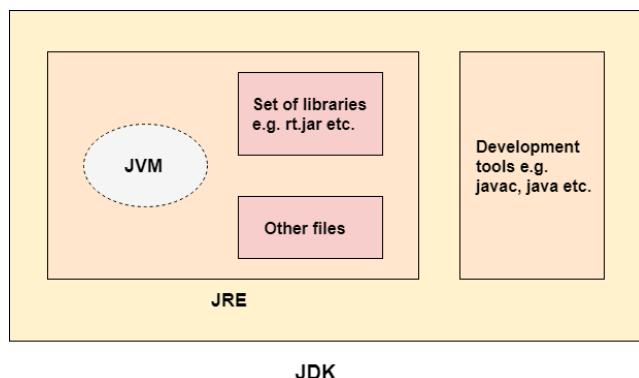


Слика 2. Позадина извршавања Јава програма [7] [8]

У односу на апликациони домен и циљану класу уређаја, Јава платформа обухвата:

1. Java SE(Standard Edition)
2. Java EE (Enterprise Edition)
3. Java ME (Micro Edition)

Java Standard Edition је развојно окружење које се састоји из две основне компоненте: JRE (Java Runtime Environment) и JDK (Java Development Kit). JRE обухвата Јава виртуелну машину (JVM) и скуп библиотека неопходних за покретање апликација написаних Јава програмским језиком. JVM се сматра кључним делом Јаве јер управо она обезбеђује платформску независност, интерпретирањем бајт кода у наредбе конкретног оперативног система. Са друге стране, JDK укључује JRE и пружа алате за развој и тестирање програма, као што су компајлери и дебагери. [5] Однос између JVM, JRE и JDK може се видети на слици 3.



Слика 3. Однос између JVM, JRE и JDK [9]

Java Enterprise Edition је заснована на Java SE и обезбеђује технологије као што су: Java Servlet, Java Server Pages, Session beans, Entity beans, JAXP и многе друге. Представља платформу која се претежно користи за развој веб апликација. Често се примењује у областима као што су интернет трговина и банкарство.

Java Micro Edition је развојно окружење за развој и извршавање апликација намењених уређајима као што су мобилни телефони и телевизори.

2.1.2. Програмски језик Јава

Јава је објектно-оријентисани програмски језик развијен од стране Џејмса Гослинга 1995. године. Сматра се једним од најпопуларнијих језика и данас има широку употребу на пољу интернет и десктоп програмирања, мобилних апликација, игара и бизнис решења. Такође, представља темељ Android оперативног система. Оно што је посебно чини моћном јесте подршка заједнице која броји десетине милиона програмера. Јава се сматра једним од најлакших програмских језика за учење.

Неки од кључних принципа на којима се заснивају су: платформска независност, објектно-оријентисаност, конкурентност, сигурност и високе перформансе.

Објектно-оријентисано програмирање (ООП) значајно олакшава пресликавање концепата из реалног света у класе нашег програма. Омогућава лакше одржавање, измене и поновну употребу кода. Неки од основних концепата ООП су: полиморфизам, наслеђивање, учаурење и апстракција.

У општем смислу, појам полиморфизам указује на нешто што има више облика. Конкретно, полиморфизам је у Јави омогућен путем наслеђивања, компатибилности објектних типова и чињенице да се методе повезују са програмом у време извршења. Преклапање метода представља још један начин постизања полиморфизма.

Наслеђивање је концепт који омогућава да подкласа наследи све атрибуте и методе надкласе коју наслеђује. На тај начин се спречава понављање истог кода и омогућава вршење измена само на једном месту, уколико за тим постоји потреба.

Учаурење је механизам којим се раздвајају особине које су јавне од особина које су скривене за друге модуле система. Постиже се тако што се атрибути у оквиру класе декларишу са `private` модификатором приступа тако да им се изван те класе може приступити само путем јавних `getter` и `setter` метода. На тај начин се постиже сакривање података као и већа контрола над њима.

Апстракција је процес којим се врши сакривање имплементационих детаља док се кориснику приказује само функционалност. У Јави постиже употребом апстрактних класа и интерфејса. Апстрактна класа је она која садржи бар једну апстрактну методу, чија се

имплементација препушта подкласи која је наслеђује. Са друге стране, интерфејс садржи само апстрактне методе. Овај механизам омогућава слабу повезаност између класа, чему треба тежити приликом развоја софтвера.

Веома значајан концепт Јава програмског језика јесте и конкурентност. Она представља способност извршавања више програма или делова програма истовремено, чиме се постижу високе перформансе. Кључни механизам за постизање конкурентности представљају нити.

2.2. GraalVM

HotSpot Јава виртуелна машина написана је програмским језиком C++ и постоји више од две деценије.[10] Може се рећи да је утемељена веома старим и комплексним кодом. Током тог периода је дошло до многих промена у свету софтверског инжењерства. У складу са тим су приметне и многе иновације у JVM екосистему. Све је више присутна тенденција великих компанија да развијају сопствене имплементације Јава виртуелне машине. Једна од њих је виртуелна машина високих перформанси GraalVM иза које стоји компанија Oracle.

Намењена је првенствено побољшању перформанси апликација написаних програмским језицима који се извршавају помоћу Јава виртуелне машине. Ту спадају: Java, Groovy, Scala, Kotlin и Clojure. Међутим, поред набројаних језика, постоји подршка и за многе друге популарне језике као што су JavaScript, Python, Ruby и остали.[11] То доводи до вишејезичности као једне од битних карактеристика GraalVM. Прецизније, она омогућава и извршавање апликација које истовремено користе више различитих програмских језика. Добар практичан пример ове функционалности би била Јава апликација која користи Python за потребе машинског учења. Такође, на тај начин се омогућава да стручњаци који познају различите програмске језике заједно сарађују на истој апликацији.

Кључна карактеристика GraalVM јесте то што знатно побољшава перформансе свих Јава и JVM блиских апликација без икакве потребе за променом кода. Централни део ове технологије представља Graal компајлер који је написан програмским језиком Јава. Он применом напредних алгоритама, техника и анализа кода производи високо оптимизован машински код који се извршава брже, смањује време потребно за garbage collecting процес и захтева много мању употребу ресурса. Пре свега, смањује време извршавања корисничких захтева. Употребом GraalVM компајлера, JVM језици постижу убрзање до 30% у односу на стандардни компајлер. Код који садржи модерније Јава карактеристике, као што су стримови и ламбда изрази, ће осетити још боље перформансе.[12] Апликације које се извршавају брже, такође брже ослобађају заузете ресурсе процесора и меморије чиме их раније стављају на располагање другим захтевима. GraalVM постиже боље перформансе апликације уз мању употребу процесора и меморије, што су циљеви којима увек треба тежити. На тај начин се директно утиче и на смањење инфраструктурних трошкова. Конкретно, са постигнутим побољшаним перформансама је могуће обрадити исту количину захтева употребом мање количине ресурса, чиме се директно смањују трошкови куповине серверских машина и додатног хардвера потребног за покретање апликација у производном окружењу.

2.2.1. Native image

Поред претходно наведеног JIT компајлирања у време извршавања програма, највећа вредност коју нуди Graal компајлер јесте могућност Ahead-Of-Time (AOT) компајлирања. У питању је процес компајлирања који се дешава пре извршавања и чији је резултат самостални, платформски специфичан извршни програм који се назива native image. У позадини извршавања овог процеса, дешава се статичка анализа бајт кода како би се утврдило које класе и методе су дохватљиве из улазне методе апликације. То значи да све класе и методе које су неопходне за извршавање морају бити познате у тренутку компајлирања. На тај начин се обезбеђује да резултујући бинарни фајл садржи све класе и библиотеке које су неопходне за моментално извршавање апликације.[11]

Бенефити коришћења ове технологије су бројни. Брзина покретања апликација се мери милисекундама јер је процес превођења у машински код већ извршен пре самог покретања, што значи да нема више потребе за радом JIT компајлера. Такође, како се меморија и процесор више не користе за компајлирање у току самог извршавања програма, употреба ових ресурса је знатно мања у поређењу са апликацијама које се покрећу помоћу Јава виртуелне машине. Поред чињенице да је native image независан од JVM, и даље су му потребне неке од функција које она нуди, као што су управљање меморијом, нитима, сакупљање смећа и многе друге. Све наведене компоненте Graal виртуелној машини обезбеђује SubstrateVM, која је такође написана програмским језиком Јава.[13] Још један бенефит АОТ компајлирања јесте то што се оптималне перформансе апликације постижу иницијално приликом покретања и константне су, у поређењу са JIT компајлирањем где је прво потребан одређени период рада апликације како би се достигао врхунац перформанси.[12]

Поред наведених предности, native image са собом носи одређене недостатке и његово генерисање није увек једноставно. Сам процес изградње може бити веома спор. Кључни проблем представља чињеница да након АОТ компајлирања није могуће учитавање новог кода у току извршавања програма, што не одговара нужно свим апликацијама. Самим тим, ни динамичко учитавање класа није могуће. Такође, отежано је коришћење механизма рефлексије које захтева додатно конфигурисање пре него што се генерише native image.[14] Битно је напоменути и да native image долази са минималним подразумеваним скупом протокола, тако да је све додатне, који нису већ укључени, потребно ручно конфигурисати. Пример за то су протоколи HTTP и HTTPS који се укључују следећим командама:

```
--enable-http, --enable-https
```

На основу наведених предности и недостатака се може извести закључак да је native image право решење у ситуацијама када су кориснику важни време покретања и

употреба ресурса, под условом да је комплетан постојећи код подложен АОТ компајлирању без већих измена. Сама чињеница да ова технологија нуди извршни програм мање величине који се моментално покреће и захтева мању количину ресурса је чини идеалном за изградњу модерних микросервисних система. Томе доприносе и владајући трендови прелаза на мање, дистрибуиране системе који нуде високу поузданост и скалабилност у могућности брзог одговора на повећану или смањену потражњу. Многи значајни радни оквири за развој микросервисних апликација су то препознали. Spring је у оквиру још увек експерименталног пројекта Spring Native омогућио подршку за компајлирање својих апликација помоћу GraalVM native image компајлера. У наставку је дат пример основних корака неопходних за изградњу и покретање Spring Boot апликација применом Spring Native пројекта: [15]

1. Проверити да ли се користи подржана Spring Boot верзија

2. Конфигурисати Maven додатке и репозиторијуме:

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <image>
            <builder>paketobuildpacks/builder:tiny</builder>
            <env>
                <BP_BOOT_NATIVE_IMAGE>1</BP_BOOT_NATIVE_IMAGE>
                <BP_BOOT_NATIVE_IMAGE_BUILD_ARGUMENTS>
                    -Dspring.native.remove-yaml-support=true
                    -Dspring.spel.ignore=true
                </BP_BOOT_NATIVE_IMAGE_BUILD_ARGUMENTS>
            </env>
        </image>
    </configuration>
</plugin>

<repositories>
    <repository>
        <id>spring-milestone</id>
        <name>Spring milestone</name>
        <url>https://repo.spring.io/milestone</url>
    </repository>
</repositories>
```

```
<pluginRepositories>
    <pluginRepository>
        <id>spring-milestone</id>
        <name>Spring milestone</name>
        <url>https://repo.spring.io/milestone</url>
    </pluginRepository>
</pluginRepositories>
```

3. Додати Spring Native GraalVM dependency

```
<dependencies>
    <dependency>
        <groupId>org.springframework.experimental</groupId>
        <artifactId>spring-graalvm-native</artifactId>
        <version>0.8.3</version>
    </dependency>
</dependencies>
```

4. Изградити апликацију следећом Maven командом:

```
mvn spring-boot:build-image
```

5. Покренути апликацију као Docker контејнер следећом командом:

```
docker run -p 8080:8080 docker.io/library/test-service:0.0.1-SNAPSHOT
```

2.3. Spring

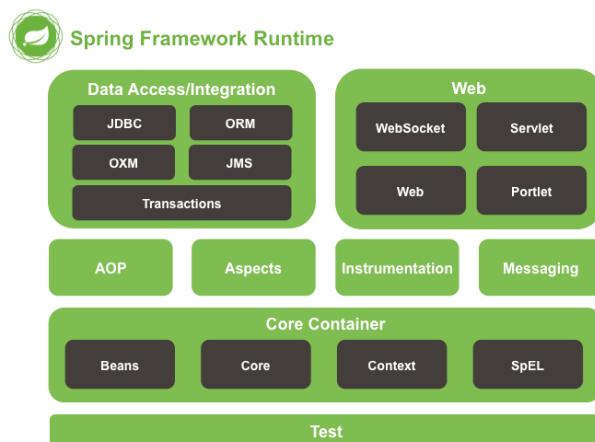
Spring представља најпопуларнији оквир за развој апликација у Јава екосистему. Настао је 2003. године, у време када се Јава ЕЕ развијала великом брзином. Може се користити у било којој Јава апликацији без обзира на њену величину или домен проблема.

Предности коришћења овог оквира су бројне. Пре свега, омогућава програмерима да се фокусирају на пословну логику апликације, без потребе за детаљним подешавањима окружења. Веома је једноставан за употребу и обезбеђује инфраструктуру за брз почетак рада на развоју апликације. Модуларне је природе, што значи да постоји могућност коришћења целог Spring оквира или само његових модула који су нам у тој ситуацији потребни. Такође, веома је значајна чињеница да иза њега стоји огромна заједница како појединачних програмера, тако и организација које га користе.

Spring обухвата следеће модуле који се могу веома једноставно изабрати и употребити у апликацији по потреби: [16]

1. Core - обезбеђује основне карактеристике као што су убрзавање зависности (енгл. Dependency Injection), валидација, аспектно-оријентисано програмирање (АОП) итд.
2. Data Access - пружа подршку за приступ подацима кроз на пример Java Persistence API и JDBC
3. Web - подржава Servlet API и Reactive API
4. Integration – подржава интеграцију кроз Java Message Service (JMS)
5. Test – пружа широку подршку за unit и интеграционо тестирање

Приказ ових модула дат је на слици 4.



Слика 4. Spring модули [17]

Inversion of Control (IoC) је један од кључних принципа Spring оквира који врши екстернализацију креирања и управљања зависностима компонената. На тај начин се

врши чишћење пословне логике од кода који укључује инстанцирање објеката и разне врсте конфигурација. Убрзавање зависности је механизам за имплементацију IoC-а и може се постићи путем конструктора, setter метода или @Autowired анотације. Наредни програмски код приказује убрзавање зависности употребом @Autowired анотације.

```
public class UserServiceImpl implements UserService {  
    @Autowired  
    UserDao userDao;  
    ...  
}
```

Конкретно, Spring у својој основи садржи контејнер који се често назива **Spring application context**, који креира компоненте апликације и управља њима. Компоненте (Spring Beans) су повезане унутар application context-a, а сам поступак повезивања се заснива на Dependency Injection принципу. Прецизније, уместо да компоненте саме креирају и воде рачуна о животном циклусу других компонената од којих зависе, препушта се контејнеру да креира и одржава све компоненте и убрзава их у оне компоненте којима су потребне.

Предност овакве архитектуре се огледа у лакшој промени између различитих имплементација, већој модуларности програма и лакшем тестирању.

Навођење контекста апликације да изврши повезивање компонената може се извршити на два начина. Први начин подразумева један или више XML фајлова који описују компоненте и њихове међусобне везе. Други начин, који је новији и чешће се употребљава, подразумева конфигурацију која се базира на Јави. Конкретно, помоћу @Configuration анотације се указује Spring-у да се ради о конфигурационој класи која обезбеђује бинове за контекст. Методе унутар ових класа се означавају @Bean анотацијом како би се указало да објекти које оне враћају треба додати у контекст као бинове. Spring помоћу скенирања аутоматски открива компоненте на classpath-у и креира их као бинове у Spring контексту апликације. [18]

Spring оквир пружа различите врсте анотација као што су: @Autowired, @Component, @Repository, @Service, @RestController, @Configuration и многе друге.

2.4. Spring Boot

Spring Boot је пројекат који је настао као надоградња Spring оквира са циљем да се програмерима олакша и убрза процес креирања Spring апликација. Он омогућава изградњу самосталних апликација са уграђеним веб сервером, спремних за продукцију уз минимално улагање напора. Може се рећи да представља најкраћи пут до покретања Spring веб апликације.

Неке од најзначајнијих карактеристика овог оквира су: [19]

1. Једноставно управљање зависностима

Spring Boot обезбеђује стартере који представљају скуп одговарајућих зависности које се лако могу укључити у апликацију. На пример, уколико желимо да креирати веб апликацију, доволно је да у pom.xml фајл додамо spring-boot-starter-web зависност која ће аутоматски укључити све библиотеке потребне за развој оваквих апликација. Наредни програмски код приказује spring-boot-starter-web зависност у оквиру pom.xml фајла.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2. Аутоматска конфигурација

Након додавања одређеног стартера, Spring Boot аутоматски конфигурише све што је потребно у апликацији. На пример, уколико укључимо spring-boot-starter-jdbc зависност, Spring Boot ће аутоматски конфигурисати bean-ове као што су DataSource, EntityManagerFactory и TransactionManager, и прочитаће детаље о конекцији из application.properties фајла.

3. Уграђен Servlet контејнер

Свака Spring Boot апликација има подразумевано уграђен веб сервер, тако да програмери не морају да се баве подешавањем Servlet контејнера и отпремањем апликације на њега. На тај начин се апликација може покренути самостално, коришћењем уграђеног сервера.

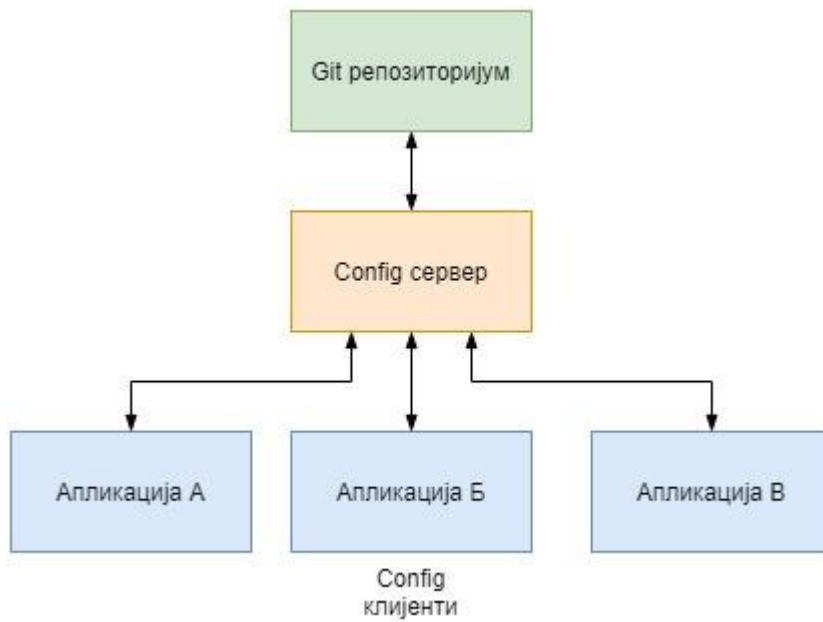
2.5. Spring Cloud

Приликом развоја дистрибуираних система често се наилази на карактеристичне проблеме везане за откривање и међусобну комуникацију микросрвиса, балансирање захтева ка различитим инстанцама, сигурност целог кластера и многе друге. Као решење је развијен Spring Cloud, модул радног оквира Spring, који пружа алате за једноставно решавање наведеног скупа проблема. Његовом применом се олакшава и убрзава развој оваквих система јер обезбеђује подршку за конфигурисање свих понављајућих али ипак неопходних параметара уз минимални утрошак напора и времена.

Веома је важна чињеница да Spring Cloud обухвата више пројеката који су усмерени на различите функционалности, па је тако могуће веома једноставно укључити у апликацију било који од њих у зависности од тренутних потреба корисника. Често је доволично само додавање жељеног стартера у пројекат како би се аутоматски конфигурисао основни скуп функционалности. Неки од кључних пројеката које Spring Cloud укључује су: Spring Cloud Config, Spring Cloud Netflix, Spring Cloud Gateway, Spring Cloud OpenFeign и многи други.[20] У наставку је дат детаљнији опис њихових улога и својстава.

2.5.1. Spring Cloud Config

Spring Cloud Config обезбеђује централизовано, екстерно место за управљање конфигурацијама целог окружења. Сама конфигурација се чува на неком од система за управљање верзијама, чији је најпознатији представник Git. У првом кораку је потребно конфигурисати апликацију која ће имати улогу config сервера. То је веома једноставна апликација која од подешавања садржи само порт на коме ради и путању ка Git репозиторијуму на коме се налазе конфигурације. Све остале конкретне апликације имају улогу config клијента. Доволично је да оне у оквиру bootstrap.yml фајла имају подешене само путању и креденцијале за приступ config серверу, који даље на основу имена клијентске апликације обезбеђује одговарајуће конфигурације које припадају клијенту и налазе се у централном Git репозиторијуму.[21] Веома је битно нагласити улогу bootstrap фајла јер се прво он учитава приликом покретања Spring апликација. Другим речима, config сервер има директну везу са екстерним конфигурационим фајловима и служи као посредник који управља конфигурацијама свих клијената који су са њим повезани. На тај начин се постиже значајна флексибилност јер је могуће променити одређене конфигурационе параметре у току рада апликације, без потребе за прекидањем, поновном изградњом и покретањем апликације. Из тог разлога се најчешће екстернализују подаци везани за конекцију са базом или messaging брокером. Описана архитектура је приказана на слици 5.



Слика 5. Spring Cloud Config архитектура

2.5.2. Spring Cloud Netflix

Spring Cloud Netflix је најпопуларнији Spring Cloud пројекат јер га чине модули усмерени на већину кључних концепата развоја дистрибуираних система. Неки од њуих су: Eureka, Ribbon и Hystrix.

Netflix Eureka је модул који се имплементира помоћу сервера и клијентских апликација које се на њега региструју. Eureka сервер има улогу централног регистра који обезбеђује податке о свим доступним сервисима, њиховим називима, адресама и броју покренутих инстанци, чиме им омогућава једноставно проналажење и међусобну комуникацију. У свим клијентским апликацијама је потребно само конфигурисати везу са Eureka сервером. На тај начин било која клијентска апликација не мора да буде везана за тачну адресу или порту друге стране са којом комуницира, јер из централног регистра по називу жељеног сервиса може добити све потребне податке. То је посебно корисно јер су у динамичном окружењу чести случајеви промена броја покренутих инстанци сервиса и њихових адреса.

Netflix Ribbon има функцију равномерног распоређивања захтева ка различитим инстанцима истог сервиса, како не би дошло до преоптерећености само једне инстанце док су друге доступне и потпуно неискоришћене. На тај начин се постиже висока ефикасност у искоришћености ресурса, мање време одзива на корисничке захтеве, као и већа поузданост целог система.

Netflix Hystrix има кључну улогу у обезбеђивању отпорности дистрибуираних система на грешке. Микросервисна архитектура се састоји од великог броја сервиса који

међусобно комуницирају, па би се грешка у једном од њих преносила и на остале што би у најгорем случају могло да доведе до пада целог система. Са бројем микросрвиса расте и сложеност решавања овог проблема. Hystrix из тог разлога омогућава дефинисање резервних метода које ће се позивати у случају да се пређе одређени праг броја поновљених грешака и спречити да се оне даље шире док се систем не опорави.[22]

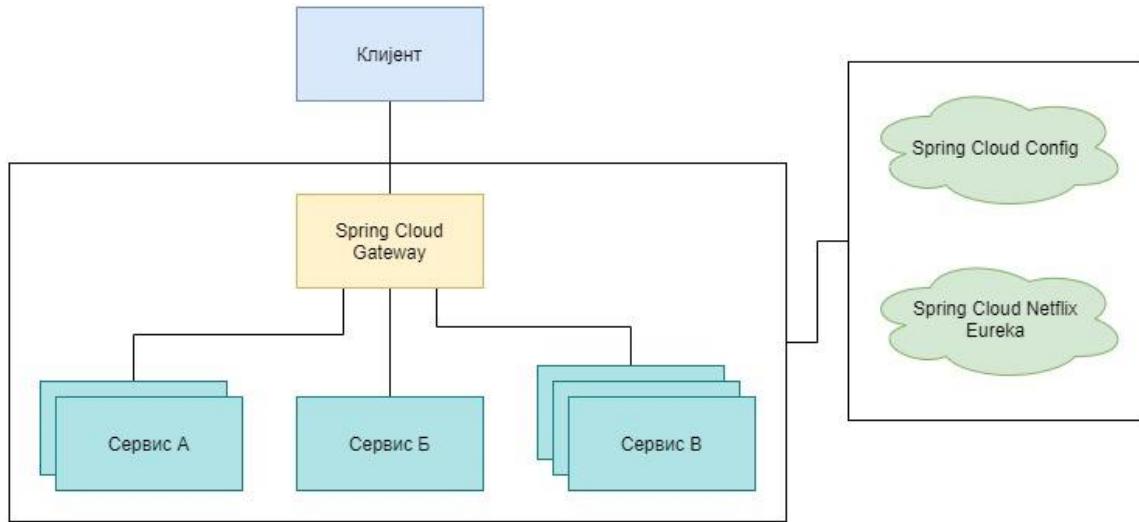
2.5.3. Spring Cloud Gateway

У микросрвисној архитектури са великим бројем сервиса би било веома тешко да клијенти комуницирају са сваким од њих појединачно. То би подразумевало да сваки клијент познаје комплетну архитектуру микросрвисног система и тачне адресе свих покренутих инстанци које се притом веома често мењају. Због тога је препоручено креирање API Gateway апликације која за клијенте има улогу јединствене приступне тачке целом систему, односно свим осталим микросрвисима. Другим речима, API Gateway представља посредника који прима захтеве од клијената и прослеђује их одговарајућем сервису који је задужен за њихово извршење. Spring Cloud Gateway је пројекат који знатно олакшава развој и конфигурисање овакве врсте апликација. Потребно је само дефинисати руте на основу којих се одређује који тачно сервис треба да изврши одређени захтев. Пожељно је користити га заједно са Netflix Eureka пројектом како би се омогућило једноставно рутирање по називу сервиса и равномерно распоређивање захтева ка њиховим инстанцама. У наставку је дат пример програмског кода потребног за конфигурисања ruta у оквиру Spring Cloud Gateway пројекта:

```
@Bean
public RouteLocator routes(RouteLocatorBuilder builder) {
    return builder.routes()
        .route(r -> r.path("/clubs/**", "/gamedays/**", "/matches/**", "/match-events/**",
"/players/**", "/performances/**").uri("lb://football-fantasy-service"))
        .route(r -> r.path("/parser/**").uri("lb://football-fantasy-parser-service"))
        .route(r -> r.path("/stats/**").uri("lb://football-fantasy-stats-service"))
        .route(r -> r.path("/leagues/**", "/teams/**", "/team-performances/**", "/team-league-
memberships/**").uri("lb://football-fantasy-team-service"))
        .route(r -> r.path("/users/**", "/login").uri("lb://football-fantasy-user-service"))
        .build();
}
```

Битно је напоменути да је Spring Cloud Gateway неблокирајућа технологија. То значи да је увек доступна нит за прихвататење захтева који се у позадини извршавају асинхроно и да не може доћи до блокирања и застоја.[23]

Пример примене различитих Spring Cloud пројеката у микросрвисној архитектури је приказан на слици 6.



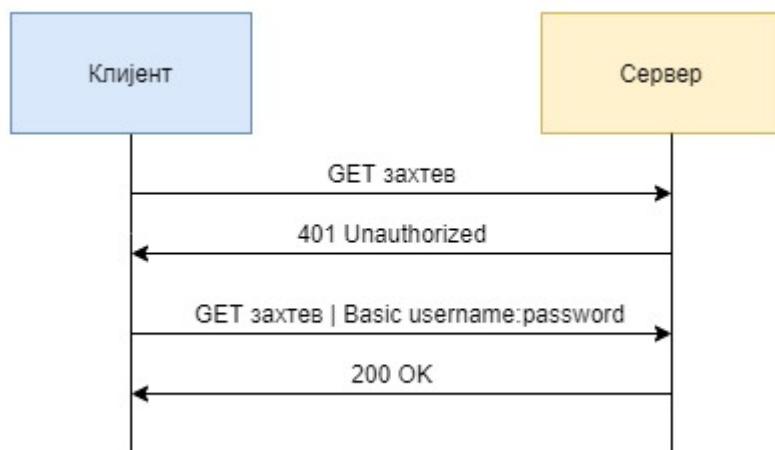
Слика 6. Spring Cloud архитектура

2.6. Spring Security

Spring Security је модул радног оквира Spring задужен за сигурност и заштиту апликација од свих познатих врста напада. Пружа могућности једноставног управљања аутентификацијом и ауторизацијом корисника система, као кључним проблемима заштите, у складу са личним потребама и захтевима.

Аутентификација је процес утврђивања идентитета корисника који приступа неком ресурсу система. Најчешће се спроводи путем корисничког имена и лозинке. Другим речима, даје одговор на питање: "Ко приступа систему?". Централно место за конфигурисање аутентификације представља метода `authenticate()` интерфејса `AuthenticationManager`, која као резултат потврђује валидност корисника или баца грешку у супротном.[24] Неки од могућих начина аутентификације које Spring Security нуди су: Basic аутентификација, аутентификација путем форме и JWT (JSON Web Token) аутентификација.

Basic аутентификација подразумева да се у оквиру сваког корисничког захтева ка серверу подеси *Authorization* заглавље чија вредност садржи корисничко име и лозинку у формату *Basic username:password*. Ово јесте најједноставније али не и најбоље решење због чињеница да је обавезно слање креденцијала у сваком захтеву, као и да није могућа одјава са система. Зато је овај приступ најприкладнији ситуацијама када се приступа екстерном API-ју. Пример захтева се може видети на слици 7.



Слика 7. Spring Security - Basic аутентификација [25]

Аутентификација путем форме представља стандард у великом броју апликација. Корисник у оквиру форме за пријављивање уноси корисничко име и лозинку и шаље POST захтев, након чега сервер врши валидацију креденцијала. Уколико је све у реду, сервер у одговору враћа идентификатор сесије, који се затим шаље у сваком наредном захтеву. Предности овог приступа су потпуна контрола над формом за пријављивање и могућност одјаве. Поступак је приказан на слици 8.



Слика 8. Spring Security - аутентификација путем форме [25]

JWT аутентификација се препоручује у ситуацијама када више различитих клијената комуницира са апликацијом. Главна предност овог приступа је то што је веома брз јер нема стање. То значи да не постоји потреба за одржавањем корисничке сесије јер су сви потребни подаци садржани у самом токену. До проблема у овом приступу може доћи уколико се открије тајни кључ за генерирање токена или уколико неко украде токен. Оваква аутентификација подразумева слање креденцијала на сервер од стране корисника, након чега сервер врши њихову валидацију. Уколико је све у реду, на серверу се генерише JWT и враћа као одговор, да би га корисник након тога слao у сваком следећем захтеву. Поступак је приказан на слици 9.



Слика 9. Spring Security - JWT аутентификација [25]

Када се корисник аутентификује, независно од претходно наведених начина, омогућен му је приступ одређеним ресурсима апликације. Међутим, у највећем броју реалних ситуација је потребно јасно дефинисати дозволе и права приступа различитим ресурсима од стране различитих врста корисника. На пример, обичан корисник не би смео да има приступ делу система намењеном администраторима. Управо томе служи

авторизација корисника. Spring Security омогућава једноставно управљање ролама и правима приступа кориснику.

Централно место за конфигурисање Spring Security модула јесте метода `configure()` у оквиру класе која наслеђује `WebSecurityConfigurerAdapter` и означена је `@EnableWebSecurity` анотацијом. На веома интуитиван начин се врши конфигурисање: начина аутентификације, метода за које је потребна аутентификација, права приступа методама по различитим ролама и дозволама, као и многих других параметара потребних за обезбеђивање сигурности апликације у целини. У наставку је дат пример кода који описује наведена подешавања:

```
@Configuration
@EnableWebSecurity
public class ApplicationSecurityConfig extends WebSecurityConfigurerAdapter {

    ...

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .cors()
            .and()
            .csrf().disable()
            .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .addFilter(new JwtAuthenticationFilter(authenticationManager()))
            .addFilter(new JwtAuthorizationFilter(authenticationManager(),
userPrincipalService))

        .authorizeRequests()
            .antMatchers("/actuator/**").permitAll()
            .antMatchers(HttpMethod.POST, "/login").permitAll()
            .antMatchers(HttpMethod.POST, "/users/register").permitAll()
            .antMatchers(HttpMethod.GET, "/clubs/all").permitAll()
            .antMatchers(HttpMethod.GET, "/clubs/parse-season-
clubs").hasAuthority("ROLE_ADMIN")
            .antMatchers(HttpMethod.POST, "/clubs/parse").hasAuthority("ROLE_ADMIN")
            .antMatchers(HttpMethod.GET, "/gameweeks/parse-season-
gameweeks").hasAuthority("ROLE_ADMIN")
            .antMatchers(HttpMethod.DELETE, "/gameweeks/**").hasAuthority("ROLE_ADMIN")
            .antMatchers("/match-events/parse-match-events/**").hasAuthority("ROLE_ADMIN")
            .antMatchers(HttpMethod.DELETE, "/match-events/**").hasAuthority("ROLE_ADMIN")
            .antMatchers(HttpMethod.DELETE, "/players/**").hasAuthority("ROLE_ADMIN")
            .antMatchers("/performances/calculate/**").hasAuthority("ROLE_ADMIN")
            .antMatchers("/team-performances/calculate/**").hasAuthority("ROLE_ADMIN")
        .anyRequest().authenticated();
    }
}
```

2.7. Hibernate

Објектно-релационо мапирање (ORM) је програмерска техника за конвертовање података између релационих база података и објектно-оријентисаних програмских језика.

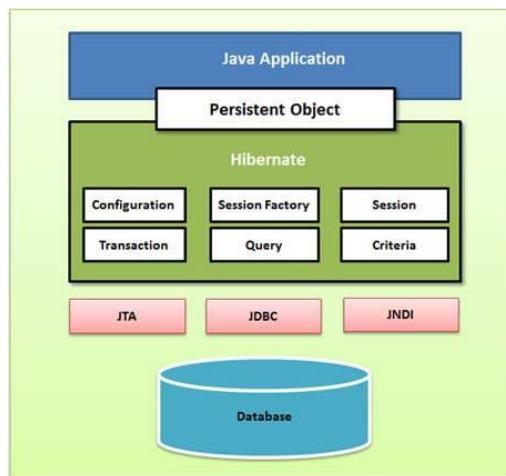
Hibernate радни оквир је један од најпознатијих алата за објектно-релационо мапирање. Одликују га веома високе перформансе у оквиру Јава апликација. Користи се за мапирање Јава класа у табеле базе података и обратно. Он обавља свак посао везан за перзистенцију објекта. За објекат се каже да је перзистентан уколико настави да постоји и након гашења програма који га је створио, односно уколико се може материјализовати и дематеријализовати. Материјализација је процес трансформације слогова базе података у објекте програма, док је дематеријализација обрнут поступак. [26]

Hibernate уместо ResultSet-а користи објекте и на тај начин подржава наслеђивање, композицију и рад са колекцијама. Користи HQL који представља објектно-оријентисану верзију SQL упитног језика. Он аутоматски генерише упите који су независни од система за управљање базом података коришћењем SQL дијалекта. Омогућава побољшавање перформанси перзистенције употребом касног учитавања (енгл. Lazy Loading) и кеширања.

Омогућава два начина за мапирање POJO (Plain old Java object) објектата:

1. XML файл
2. Јава анотације

Архитектуру Hibernate оквира чине следећи елементи: Configuration, Session Factory, Session, Transaction, Query и Criteria. [27] Детаљан приказ је дат на слици 10.



Слика 10. Детаљан приказ Hibernate архитектуре [27]

Configuration – Представља објекат који пружа два кључне компоненте које Hibernate захтева:

1. Конекција са базом података – подешава се путем једног или више конфигурационих фајлова, као што су hibernate.properties и hibernate.cfg.xml
2. Подешавање мапирања класа – ова компонента креира конекцију између Јава класа и табела базе података

Session Factory – Креира се од стране Configuration објекта, обично приликом покретања апликације. За сваку базу података у апликацији потребно је креирати посебну SessionFactory инстанцу. Кључна улога се огледа у креирању Session инстанци и снабдевању свих нити у апликацији објектима сесије. [28]

Session – Користи се за физичку конекцију са базом података. Животни циклус сесије се везује за почетак и крај логичке трансакције. Има кључну улогу да обезбеди CRUD (create, read, update, delete) операције за објекте мапираних класа. Објекти могу постојати у једном од следећа три стања [26]:

1. Транзијентно – Објекат је креiran коришћењем new оператора и још увек није повезан са Hibernate сесијом. Такође, објекат нема своју перзистентну репрезентацију у бази података и још увек му није додељен идентификатор.
2. Перзистентно – Објекат је повезан са Hibernate сесијом претходним снимањем или учитавањем из базе. Он има репрезентацију у бази података и постављену вредност идентификатора. У овом стању Hibernate аутоматски синхронизује стање објекта са стањем у бази података по завршетку трансакције.
3. Одвојено – Објекат је био у перзистентном стању али је у међувремену затворен објекат сесије за који је био повезан. Поново прелази у перзистентно стање када се повеже са новим објектом сесије.

Такође, врши креирање Transaction, Query и Criteria објеката.

Transaction – Обезбеђује методе за управљање трансакцијама и није обавезан за употребу у оквиру Hibernate апликације.

Query – Користи SQL или HQL (Hibernate Query Language) за комуникацију са базом података. Такође, користи се за постављање параметара упита и за извршавање упита.

Criteria – Користи се за креирање и извршавање објектно-оријентисаних упита

2.8. Spring Data JPA

На самом почетку, потребно је објаснити разлику између Hibernate, JPA и Spring Data JPA технологија. JPA (Java Persistence API) представља спецификацију за перзистенцију и управљање подацима, као и за мапирање између Java објекта и табела релационих база података. Hibernate је алат који представља имплементацију наведене спецификације. Са друге стране, Spring Data JPA је део Spring Data модула који није имплементација JPA, већ представља додатну апстракцију која значајно смањује количину стандардног, понављајућег кода потребног за комуникацију са базом података. Другим речима, то је додатни слој који подразумева да испод њега ради нека од JPA имплементација као што је Hibernate.[29] Највећа предност овог Spring пројекта јесте то што значајно смањује напоре потребне за извршавање једноставних упита над базом података.

У самом коду, пре свега треба подесити параметре за конекцију на основу којих ће EntityManagerFactory извршити повезивање са базом података. Затим треба обележити одговарајућим Јава анотацијама доменске класе које ће се перзистирати, како би се омогућило објектно-релационо мапирање. На следећем примеру је означенено да ће се класа користити у оквиру контекста анотацијом @Entity, повезана је са називом табеле базе података анотацијом @Table, одређен је идентификатор класе и стратегија његовог генерисања анотацијама @Id и @GeneratedValue, повезани су одговарајући атрибути класе са колонама табеле и дефинисане су врсте релација између класа анотацијом као што је @OneToMany. Поред ње, веома су битне и анотације @ManyToOne и @OneToOne.

```
@Entity
@Table(name = "clubs")
public class Club implements Serializable {

    private static final long serialVersionUID = 7705085158586004971L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "created_on")
    private LocalDateTime createdOn;
    @Column(name = "modified_on")
    private LocalDateTime modifiedOn;

    private String url;
    private String name;
    private String image;
    private String manager;
    @OneToMany(mappedBy = "club", cascade = CascadeType.REMOVE, fetch = FetchType.EAGER)
    private List<Player> players;

}
```

Након тога је доволјно само направити интерфејс у слоју приступа подацима који ће наследити постојећи JpaRepository интерфејс. Он аутоматски обезбеђује основне CRUD операције над базом података као што су: `save`, `findOne`, `findAll`, `count`, `delete` и многе

друге. Поред наведених обезбеђених операција, олакшано је и дефинисање сопствених метода. На основу кључних речи у називима метода као што су findBy, orderBy, and, or и многе друге, које се користе у комбинацији са називима атрибута, Spring Data JPA аутоматски генерише одговарајуће упите па нема потребе самостално писати имплементацију таквих метода. Поред тога, могуће је и самостално писање JPQL или изворних SQL упита у оквиру @Query анотације. Пример кода једног таквог интерфејса дат је у наставку:

```
public interface ClubRepository extends JpaRepository<Club, Long> {  
    Club findByUrl(String url);  
  
    @Query(value = "select name from clubs", nativeQuery = true)  
    List<String> findAllNames();  
}
```

Такође, обезбеђена је подршка за аутоматску пагинацију и сортирање која се добија наслеђивањем *PagingAndSortingRepository* интерфејса. Олакшано је и управљање трансакцијама једноставним коришћењем анотације @Transactional на нивоу целе класе или на нивоу појединачних метода.

2.9. JUnit

Софтверски тест представља део програма који извршава неки други део програма и процењује да ли он резултује очекиваним стањем и понашањем.[30] Он помаже програмеру да верификује исправност логике одређеног дела програма и обезбеђује одређени ниво сигурности да програм ради оно што је потребно.

JUnit је радни оквир за тестирање Јава апликација. Он представља стандард међу алатима за тестирање. JUnit тест је метода која се налази у оквиру класе означене @SpringBootTest анотацијом и намењена је само тестирању. Да би дефинисали одређену методу као тест методу, потребно је означити је @Test анотацијом. Према конвенцији, назив тест класе треба да буде исти као назив класе чије се методе тестирају, уз додатак речи „Test“ на крају. За проверу очекиваног и стварног резултата углавном се користи assertEquals() метода, која као параметре прима очекивани и стварни резултат. Ако су ове две вредности једнаке, тест је прошао. Наредни програмски код представља пример тест класе:

```
@SpringBootTest
@ActiveProfiles("test")
@Transactional
class TeamServiceTest {

    @Autowired
    TeamService teamService;
    @Autowired
    TeamSetup teamSetup;
    Team team;

    @BeforeEach
    void beforeEach() {
        team = teamSetup.getFullValidSetup();
    }

    @Test
    void testSave_InvalidTeamSize() {
        Log.info("Test save team with invalid size");
        List<TeamPlayer> players = team.getTeamPlayers();
        players.remove(0);

        Exception exception = assertThrows(TeamException.class, () -> {
            teamService.save(team);
        });
        assertEquals("[TEAM_NOT_VALID] Invalid team size!", exception.getMessage());
    }

    @Test
    void testSave_InvalidTeamDistinctPlayers() {
        Log.info("Test save team with duplicate players");
        List<TeamPlayer> players = team.getTeamPlayers();
        players.remove(0);
        players.add(players.get(0));

        Exception exception = assertThrows(TeamException.class, () -> {
            teamService.save(team);
        });
        assertEquals("[TEAM_NOT_VALID] Team contains duplicate players!",
            exception.getMessage());
    }
}
```

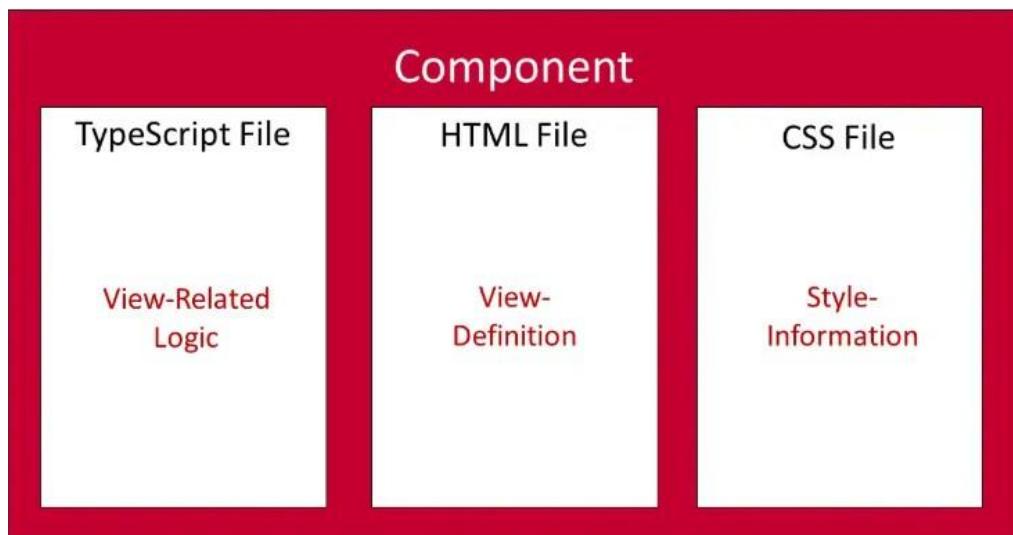
2.10. Angular

Angular је један од најпопуларнијих радних оквира за креирање веб апликација базиран на TypeScript програмском језику. Развијен је и вођен од стране Google-а, док иза њега постоји огромна заједница појединача и организација које га користе. Неке од најпознатијих компанија које користе Angular су: Google, Apple, Microsoft, Udemy, YouTube.[31]

Користи се за једноставно креирање апликација са једном страницом (single-page applications). То су апликације које учитавају само једну HTML страну чији се садржај динамички мења интеракцијом корисника са апликацијом, без новог учитавања. Angular представља и врсту софтверског патерна јер његове конвенције помажу да се пројекат изгради на одржив начин. Такође, овај оквир омогућава давање динамичког понашања HTML елементима. Овај оквир обезбеђује веома битан концепт, познат под називом повезивање података (data binding). Промене модела тј. података узрокују аутоматско ажурирање погледа (view) и обрнуто.

Angular апликације су организоване на основу модула. **Модул** представља механизам заједничког паковања компоненти, сервиса и осталих елемената апликације. На тај начин се они могу једноставно користити у различитим деловима апликације.

Angular компоненте представљају мање градивне блокове у оквиру апликације, које обухватају следећа три фајла: TypeScript, HTML и CSS фајл. Структура компоненте може се видети на слици 11.



Слика 11. Структура Angular компоненте [32]

Angular сервиси служе за централизацију пословне логике и њено потпуно одвајање од погледа. Захваљујући убрзавању зависности, они се могу једноставно користити у било ком делу апликације.

Временом, развијени су веома корисни алати Angular-Cli и Angular Material, који пружају могућности као што су брзо генерисање пројекта и компоненте корисничког интерфејса које омогућавају једноставнију интеракцију корисника и система. Ови алати нису укључени у језгро Angular-а и у потпуности су опционали.

2.10.1. Angular-Cli

CLI је скраћеница за интерфејс командне линије (command line interface). Angular Cli баш то и представља, алат који се користи путем командне линије. Предуслов за његову инсталацију је да на машини постоје Node.js и npm.

Најчешће се користи за генерисање новог Angular пројекта, који се састоји од огромног броја фајлова. Њихово ручно креирање на почетку сваког новог пројекта би захтевало превише труда и времена. Тај проблем решава Angular-Cli, генерисањем свих тих фајлова уместо програмера, једноставном командом:

```
ng new naziv-projekta
```

По завршетку извршавања ове команде, добијамо готов пројекат који је спреман за покретање и садржи све што нам треба за почетак рада. Поред тога, Angular-Cli се користи за генерисање свих кључних елемената Angular апликације, као што су: класе, компоненте, модули и сервиси. То се може постићи следећим командама:

```
ng g component naziv-komponente  
ng g module naziv-modula  
ng g service naziv-servisa
```

Као резултат извршавања ових команда креирани су фајлови који садрже неопходан подразумевани код. На тај начин се смањује понављајући посао и постиже се значајна уштеда времена.

Angular-Cli се користи и за покретање апликације командом:

```
ng serve
```

2.10.2. Angular Material

Angular Material је библиотека за компоненте корисничког интерфејса. Компоненте ове библиотеке помажу у изградњи и улепшавању веб страница и апликације у целини, поштујући принципе као што су портабилност и респонзивност.

Пример програмског кода који креира Angular Material дугме:

```
<button mat-button color="primary"> </button>
```

2.10.3. TypeScript

Као што је већ речено, Angular се базира на TypeScript програмском језику. Он је развијен од стране Мајкрософта и представља надоградњу JavaScript језика, коме додаје објектну оријентисаност и могућност статичке типизације.

2.11. Elastic Stack

Elastic Stack је једна од најпопуларнијих платформи за централизовану анализу логова и мониторинг система, коју заједно чине следеће технологије: Beats, Logstash, Elasticsearch и Kibana.

Beats укључује агенте који се инсталирају на серверским машинама ради сакупљања и прослеђивања логова. Њихова предност је то што захтевају готово неприметну количину ресурса, чиме се спречава да њихов рад успорава производне сервисе. Beats фамилија обухвата више различитих врста агената (енгл. shippers) као што су: Filebeat, Metricbeat, Packetbeat, Winglobeat и други.[33] Најпознатији је Filebeat чија је улога прикупљање логова из фајлова на дефинисаним путањама. Metricbeat је такође значајан јер има улогу прикупљања системских логова и метрика.

Logstash је агрегатор који има улогу прикупљања, трансформисања и даљег прослеђивања логова, најчешће ка Elasticsearch бази података. Конфигурациони фајл *logstash.conf* састоји од три блока:

- Input – Помоћу различитих input додатака (енгл. plugins) дефинише изворе са којих ће се прикупљати подаци. Ту спадају: Beats, Kafka, JMS, JDBC и многи други.
- Filter – Блок у оквиру којег се врши трансформисање логова уколико је то потребно. Могуће је додати или уклонити одређена поља, променити њихов формат и слично.
- Output – Дефинише дестинације на које се логови даље прослеђују помоћу различитих output додатака. Најчешће је следећа дестинација Elasticsearch.

У наставку је дат пример једноставне Logstash конфигурације:

```
input {  
    beats {  
        port => "5044"  
        codec => "json"  
        type => "logback"  
    }  
}  
  
filter {  
}  
  
output {  
    elasticsearch {  
        hosts => ["http://localhost:9200"]  
        index => "logstash-%{+YYYY.MM.dd}"  
    }  
}
```

Elasticsearch је срце целе платформе јер има кључну улогу индексирања и складиштења података. У питању је NoSQL база података коју одликују снажне могућности анализе и претраге података. У оквиру ове базе се документи чувају у оквиру индекса, као неструктурирани JSON објекти, при чему се могу и дефинисати мапирања типова како би се побољшале перформансе претраживања.

Kibana је технологија презентационог слоја која ради над подацима складиштеним у Elasticsearch бази. Пружа веома једноставан и интуитиван кориснички интерфејс за анализу и визуелизацију наведених података, без потребе за самосталним писањем упита.

Бенефити које нуди Elastic Stack су поготово изражени у данашњим микросрвисним системима, где је готово немогуће пратити апликативне логове свих сервиса појединачно. Тај проблем се решава централизовањем свих логова, на основу којих се на једном месту добија увид у перформансе целог система као и могућност предвиђања будућих проблема. Самим тим је омогућено проактивно деловање како би се спречило да дође до проблема који би потенцијално могли да угрозе пословање и начине већу штету. Са тим у вези, Elastic Stack нуди и могућност узбуњивања уколико се уочи нека правилност у јављању грешака, где корисник добија обавештење путем мејла или неког другог дефинисаног канала.

Препоручена Elastic Stack архитектура је приказана на слици 12.



Слика 12. Elastic Stack архитектура [34]

У сложенијим системима који генеришу већу количину логова се препоручује додавање Kafka брокера између Beats и Logstash компоненти, који има улогу бафера у случају преоптерећености. Наведена архитектура је приказана на слици 13.



Слика 13. Elastic Stack архитектура у сложеним системима [34]

3. Преглед карактеристика монолитне и микросервисне архитектуре

На највишем нивоу апстракције, архитектура система описује његове кључне компоненте, као и начин на који су оне међусобно повезане.[1] Једна од првих великих одлука које треба донети приликом развоја софтверског система је најчешће везана за његову архитектуру. Добро постављена архитектура дугорочно има значајан утицај на успех система у целини, његове перформансе, квалитет и лакоћу одржавања. У раној фази развоја отклања ризике који би у будућности потенцијално могли да прерасту у веће проблеме, скупе и компликоване за превазилажење. Квалитетном архитектуром се сматра она која може ефикасно да одговори на све будуће потребе и циљеве организације. Она представља темељ на коме је изграђен комплетан софтвер али и путоказ у развоју сваке нове компоненте система. Суштина софтверске архитектуре је излагање структуре система, са контролисаним приказом имплементационих детаља.

Добре одлуке везане за архитектуру система потребно је донети проценом предности и недостатака примене одређених решења у специфичном контексту, а не на основу општих трендова. [35]

Са еволуцијом софтверског инжењерства упоредно је као дисциплина расла и софтверска архитектура. Одговарањем на промене које је ново време доносило мењале су се технике, стилови и правци размишљања везани за архитектуру. Са тим у вези, често се у свету софтверског инжењерства наилази на два кључна термина која описују архитектуру: монолити и микросервиси.

3.1. Монолитна архитектура

Претходне генерације софтверских компанија су се ослањале искључиво на чисту монолитну архитектуру апликација. Монолитна апликација је јединствени извршни програм и представља традиционалан избор у развоју софтверских система. Другим речима, то је систем чије све функционалности морају заједно да се покрену. Пример монолитног система је приказан на слици 14.



Слика 14. Монолитна архитектура

Првобитно су то биле јединствене, недељиве јединице које обједињују све функционалности и ресурсе на једном месту. Данас се и велике монолитне апликације могу поделити на више модула при чему највећи изазов представља дефинисање граница између њих. Што више људи ради на једном месту, веће су шансе да се једни другима нађу на путу. Често долази до ситуација да више програмера жели да изврши измене на истом делу кода, као и неспоразума везаних за питање одговорности над одређеним деловима апликације и доношење одлука. Добро постављене границе дају могућност поделе посла унутар тима по модулима, чиме се избегавају конфликти у раду на истом делу кода и омогућава да више људи паралелно ради на различитим модулима апликације. На тај начин се постиже и јасна подела одговорности у ситуацијама када је потребно направити одређене измене или додати нову функционалност. Модуларни монолит се у пракси показао као добро решење за компанију Shopify.[36] Приликом разматрања питања архитектуре система, корисно је узети у обзир и овај стил јер успешно превазилази неке од недостатака чисте монолитне а и микросервисне архитектуре. Пример монолитне апликације подељене на модуле је дат на слици 15.



Слика 15. Модуларни монолит

Погодне су када постоји потреба за мањом апликацијом, без сложене пословне логике и флексибилности, или када се ради о раним фазама пројекта. Већина данашњих успешних апликација су започете као монолити. Представљају прави избор када се ради о новој пословној идеји коју је потребно што пре спровести у дело и валидирати њену вредност. Развијају се брже, једноставније и сматра се да су погодније за одржавање све док функционалности не постану сувише сложене. То је посебно битно у ситуацијама када на развоју система ради само мањи тим програмера. Све то их чини идеалним за примену у стартап компанијама, где у почетним фазама долази до честих експериментисања и промена правца којима пројекат иде тако да се визија финалног производа може знатно разликовати од оне која је замишљена на самом почетку.

Како се комплетан код налази на једном месту, олакшана је његова поновна употребљивост за разлику од микросрвисне архитектуре која захтева копирање или издвајање дељеног кода у посебну библиотеку. Са тим у вези, и покретање end-to-end тестова је доста брже и лакше.[3] Монолитне апликације су једноставније за дебаговање и тестирање. Из истог разлога су потребни мањи напори када су у питању поступци који утичу на апликацију у целини, као што су логовање, кеширање, надгледање перформанси, сигурност и слично. Докле год је ово стандардни начин развоја апликација, сваки тим ће поседовати потребна знања за њихову изградњу.

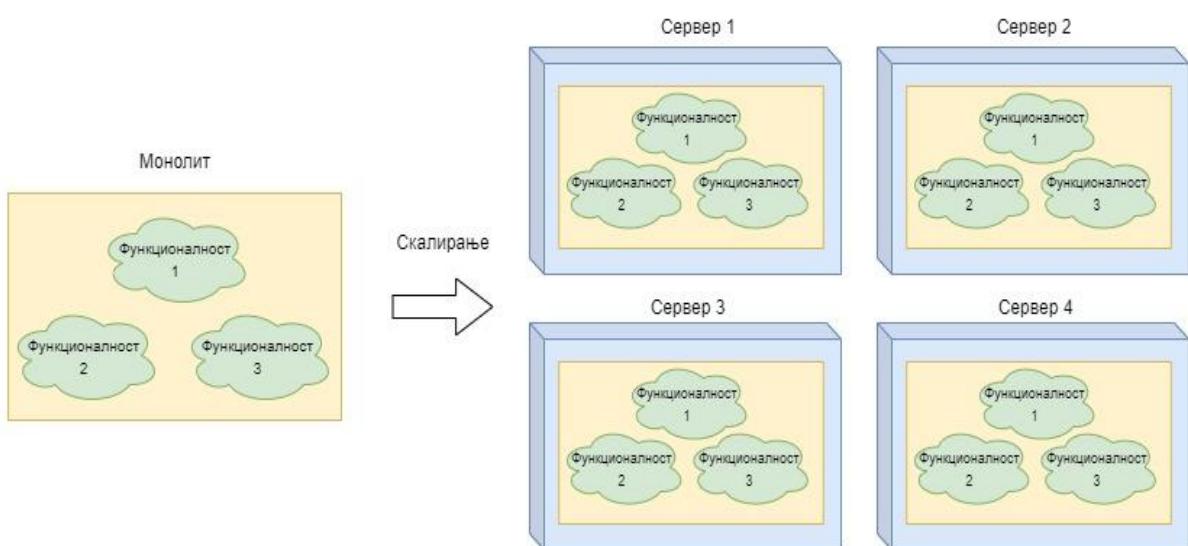
Упркос томе, популарност монолитне архитектуре опада због разних проблема које доноси са собом. Они се пре свега односе на могућности скалирања, брзину вршења измена, превелику количину кода на једном месту и примену нових технологија.

Овај приступ има лимите у виду величине и сложености апликација. Са порастом сложености, монолитни системи постају тешки за разумевање. То знатно смањује могућности вршења брзих, исправних измена и надоградњи система. Укључивање нових програмера у рад на постојећем пројекту са превеликом базом кода изискује много више

времена за упознавање. Неконтролисан раст апликације најчешће доводи до појављивања анти патерна у коду и утиче на пад његовог квалитета. Такође, велике апликације оптерећују развојно окружење и потребно је више времена за њихово покретање што директно утиче на смањење продуктивности и брзине рада програмера.[37]

Повезаност представља меру зависности одређене компоненте од других компонената система. Другим речима, то је немогућност компоненте да изврши своју функцију уколико не постоји нека друга компонента.[38] Због тога треба тежити избегавању повезаности свих врста у развоју софтвера. Монолитну архитектуру одликује висок ниво повезаности па је неопходно да све повезане компоненте буду присутне како би се програм извршио успешно. Веома је тешко имплементирати измене у великим и комплексним монолитним апликацијама са високим степеном повезаности јер свака измена кода може утицати на систем у целини. То доводи до честих мануелних тестирања комплетног система. Наведене чињенице знатно отежавају одржавање система. Свака измена захтева поновно покретање целог система. Још један од проблема монолитне архитектуре представља поузданост. Грешка у било ком делу апликације потенцијално може да наруши функционисање целог система.

Скалирање монолитних апликација се врши по принципу „све или ништа“. То значи да није могуће независно скалирање појединачних компоненти система, већ само скалирање апликације у целини. Када се повећају потребе за одређеним функционалностима, једино је могуће повећање ресурса за целу апликацију, а не само за појединачне функционалности. На тај начин се стварају велики, непотребни трошкови за потребе хардвера чији је узрок чињеница да компоненте система нису изоловане међу собом. На слици 16 је дат пример скалирања монолитне апликације.



Слика 16. Скалирање монолитне апликације

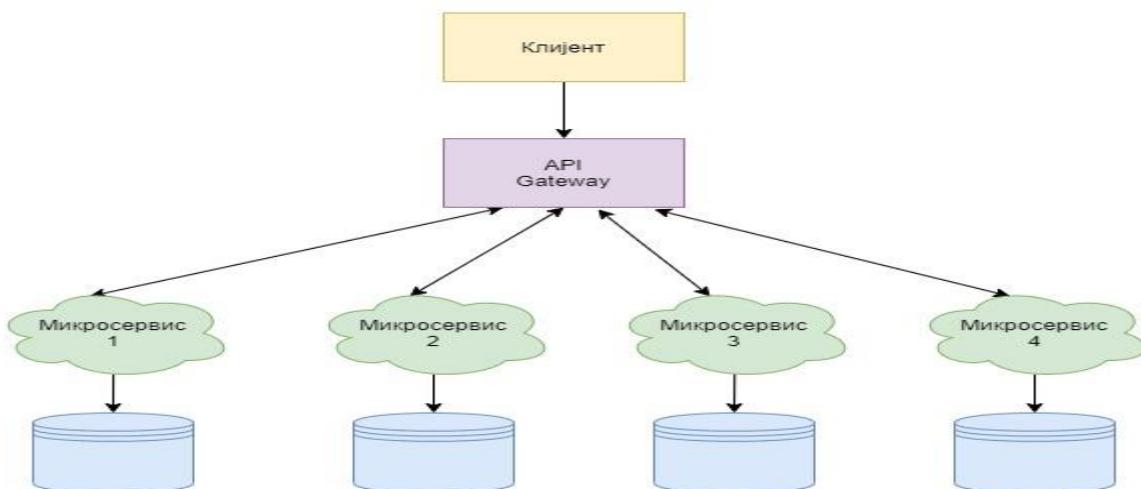
Природа монолитне архитектуре је таква да не подржава концепт честих измена и надоградњи система. Свака, чак и минимална промена, захтева поновно компајлирање модула у коме је настала и покретање комплетног система. Из тог разлога се често дешава одлагање појединих измена у производном окружењу, што доводи до проблема некомпетентности у данашњим условима пословања које захтева брзо и ефикасно прилагођавање свим тржишним променама.

Монолитне апликације нису погодне за прихватавање нових технологија. Промена радних оквира или програмских језика има утицај на цео систем што захтева утрошак велике количине времена за поновни развој апликације, па се може сматрати веома ризичном одлуком. Почетни избор технологија је најчешће дугорочан и веома тешко се мења. У великим системима, чак и промена верзије одређених технологија може довести до проблема некомпатибилности унутар система.[39]

Све наведено је довело до појаве нездадовољства корисника и смањења популарности монолитне архитектуре у модерном развоју софтверских система. Новонастали, модерни захтеви тржишта, жеље клијената и остали фактори захтевали су темељно редизајнирање традиционалних метода монолитних апликација. Као решење се појавила микросервисна архитектура.

3.2. Микросервисна архитектура

Сем Њуман дефинише микросервисе на следећи начин: „Микросервиси су мали сервиси који раде заједно“.[1] Основна идеја која стоји иза ове архитектуре јесте креирање мањих, индивидуално управљивих јединица које су међусобно повезане. Сваки микросервис је мала апликација која има сопствену пословну логику, базу података и конкретну функцију коју обавља. Најопштије речено, ова архитектура подразумева развој једне апликације преко скупа мањих, независних, децентрализованих сервиса.[39] Микросервис као део ширег система може користити функционалности других сервиса или учинити доступним своје функционалности остатку система. Примена ове архитектуре није препоручљива за променљиве, недефинисане системе, већ се показује као добро решење тек након што је успешно постављен темељ пројекта. На слици 17 је представљен пример микросервисног система.



Слика 17. Микросервисна архитектура

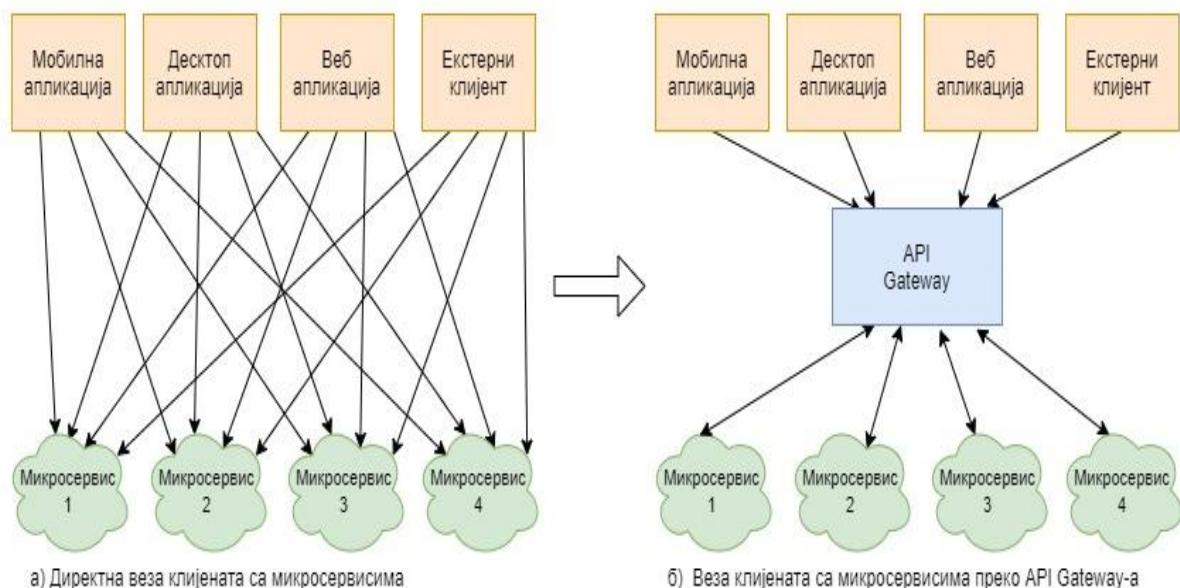
Неки од резултата испитивања у којем је учествовало неколико стотина сениор програмера, наведених у оквиру извештаја „Global Microservices Trends“ компаније Lightstep су следећи: [40]

- „91% испитаника користи или планира да користи микросервисе“
- „86% испитаника очекује да микросервиси постану подразумевани у року од пет година“
- „Кључни мотиви за усвајање микросервиса су повећана агилност (82%) и скалабилност (78%)“
- „99% испитаника пријављује изазовност коришћења микросервиса“

Једно од најчешћих питања везаних за микросервисну архитектуру се односи на величину микросервиса, која представља веома апстрактан појам. Оно што је сигурно јесте да се њихова величина не мери линијама кода. Документ (енгл. White paper)

„Building and deploying microservice applications” наводи да се префикс „микро” односи на опсег функционалности микросрвиса, а не количину његовог кода”.[35] Chris Richardson сматра да је циљ да интерфејс микросрвиса буде што је могуће мањи.[36] На микросрвисну архитектуру се може применити принцип једне одговорности по коме је микросрвис задужен за само један посао и требало би да постоји само један разлог за његову промену. Циљ је да микросрвис буде компонента система коју је могуће заменити или унапредити независно од остатка система.[39]

За комуникацију између микросрвиса се најчешће користи REST API као вид синхроне комуникације или API заснован на порукама за асинхрону комуникацију. Екстерни корисници као што су мобилне, десктоп или веб апликације, немају директан приступ овим сервисима већ се за њих излаже такозвани API Gateway као централно место приступа систему. У случају да ова компонента не постоји, сваки клијент би морао да зна тачну адресу на којој је покренут сваки микросрвис система. Ако се узме у обзир да се у пракси најчешће покреће већи број инстанци сваког сервиса као и да су њихове адресе променљиве јасно је да је Gateway неизоставни елемент микросрвисне архитектуре. Његова примарна функција је рутирање захтева, односно да прихвати захтев од корисника и проследи га сервису који је одговоран за његово извршење. Може се рећи да представља врсту фасаде микросрвисног система. Представља идеално место за имплементацију функционалности које су заједничке за све сервисе као што су балансирање терета, сигурност, мониторинг и контрола приступа. На слици 18 а) се могу видети позиви клијената ка систему када не постоји Gateway, док слика 18 б) приказује решење овог проблема додавањем Gateway-а. Са растом броја микросрвиса приказани проблем дугорочно постаје неодржив.

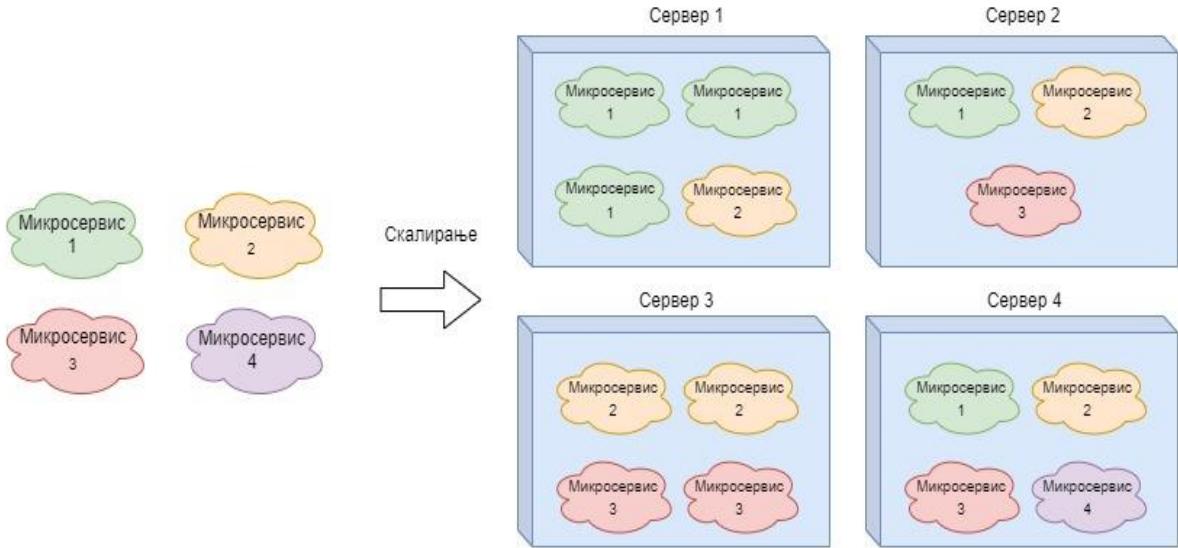


Слика 18. API Gateway

Следећа битна компонента микросервисне архитектуре јесте централни регистар сервиса. Због природе ове архитектуре која подразумева често додавање, покретање, гашење нових сервиса и њихових инстанци, сви сервиси морају да буду свесни једни других. Приликом покретања сервиса је потребно да се он пријави на централни регистар са именом, адресом и портом на коме ради. На тај начин ће Gateway моћи преко имена да проследи захтев одговарајућем сервису и ако се промени његова адреса. Такође, моћи ће да врши и равномерно распоређивање терета јер је свестан постојања свих инстанци одређеног сервиса.

Микросервисна архитектура изнад свега нуди флексибилност и велике могућности избора у сваком смислу, из којих произилазе све остале предности овог приступа. Кључни предуслови за то су добро дефинисане границе и слаба повезаност између микросервиса. Слаба повезаност постоји када је могуће направити одређену измену која неће захтевати измене на другим местима. Над независним сервисима се могу вршити измене које неће утицати на остатак система, чиме се смањује могућност појаве непредвиђених проблема.

Слаба повезаност између микросервиса доноси бројне предности као што су независно вршење измена, скалирање и покретање сервиса. Уколико се деси грешка у једном микросервису, она ће бити изолована и неће утицати на остале функционалности система које ће и даље бити доступне кориснику. Измена једног микросервиса захтева само његово поновно покретање, без потребе за координацијом са осталим тимовима и чекањем осталих сервиса. То знатно доприноси флексибилности система, као и уштеди времена. Уколико се уочи да за одређену функционалност постоји већа потреба и потражња могуће је извршити скалирање тако што ће се додатни ресурси дodelити само одговарајућем микросервису који је за њу задужен. На тај начин се такође постиже значајна уштеда новца. Ако се занемаре све друге предности микросервисне архитектуре, концепт независног развоја, покретања и скалирања сам по себи пружа изузетне бенефите. Пример скалирања микросервиса је приказан на слици 19.



Слика 19. Скалирање микросрвиса

Из структуре система би требало произвести копију тимске структуре организације. Микросрвисној архитектури одговарају мултифункционални тимови састављени од стручњака широког опсега знања, организовани око сервиса који су им додељени. Овакви тимови су високо продуктивни и одговорни су за комплетан животни век и све аспекте својих сервиса, а не само за њихов развој. На тај начин су тимови увек близко упознати са радом њихових сервиса у производном окружењу. Увођење нових функционалности које захтевају измене у сваком од слојева апликације, од базе података до корисничког интерфејса, се врше ефикасније када је за то задужен само један тим. Избегава се непотребан утрошак времена на комуникацију између тимова и спречава истовремени рад више тимова на истом делу кода. Такође, смањује се потреба да се за имплементацију неке функционалности чека да други тим уради свој део посла, што је честа ситуација у пракси.

Уместо дељења једне базе података, неопходно је да сваки сервис поседује своју посебну базу јер је то предуслов независности микросрвиса. То може довести до дуплирања појединих података, међутим такав приступ је неопходан уколико корисник жели да оствари све бенефите коришћења ове архитектуре. Пре свега јер се на тај начин постиже слаба повезаност, а затим и независност у виду избора базе података. Спречава се ситуација у којој се једна измена базе пресликава на већи број сервиса. Овај приступ омогућава различитим микросрвисима да исте ентитете чувају у облику који њима одговара. Сваки микросрвис може да користи тип базе који највише одговара његовим потребама, а да то не утиче на остале сервисе. Тако је у пракси чест случај комбиноване употребе релационих и нерелационих база података.

Уколико се нови програмер прикључи пројекту који је у развоју, неће му требати пуно времена да се прилагоди јер је потребно да разуме само функционалности појединачних

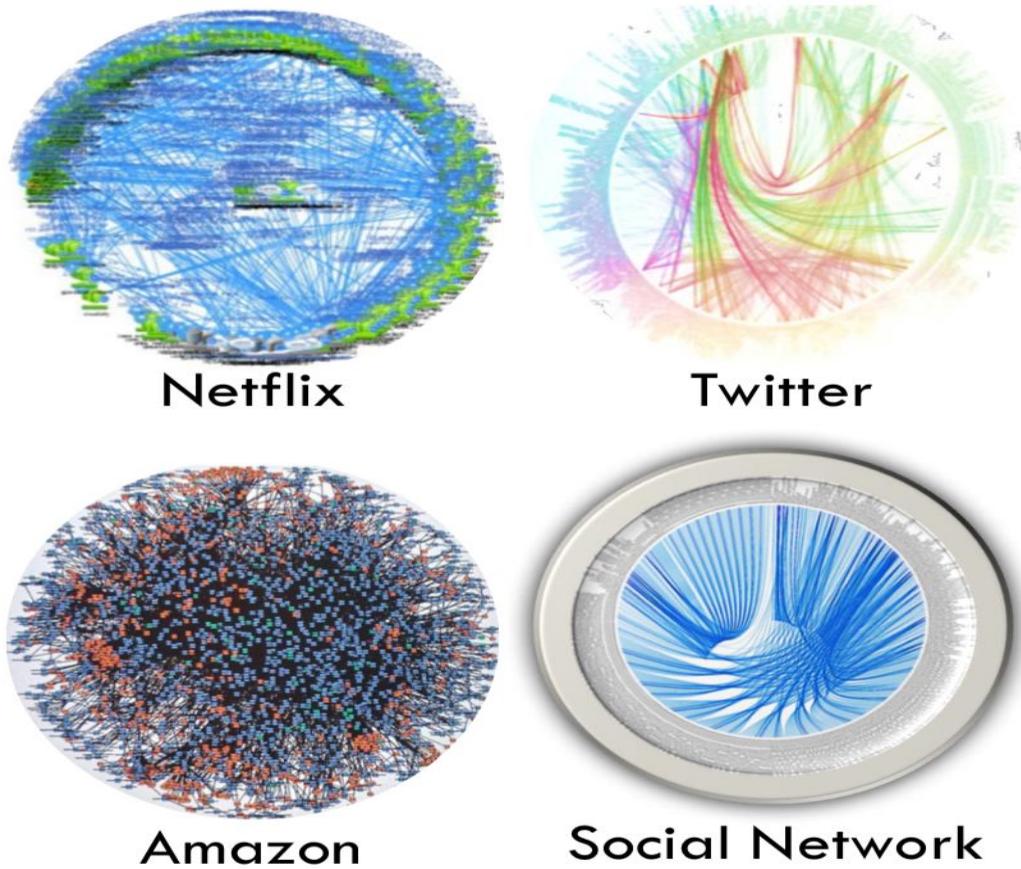
микросервиса на којима ради а не система у целини. Такође, ова архитектура знатно олакшава поделу посла. Сваки тим може независно да ради на сервису за који је задужен. Такође, мање компоненте су једноставније за развој, разумевање и одржавање. Самим тим је олакшано додавање нових функционалности и њихово тестирање. Величина микросервиса утиче и на повећану продуктивност програмера јер се апликације брже покрећу и не оптерећују претерано развојно окружење.

Још једна од веома битних карактеристика микросервисне архитектуре јесте флексибилност у избору технологија. Програмери имају слободу да изаберу технологију која највише одговара њиховом сервису, без обзира на технологије које су се користиле пре тога или технологије које користе остали сервиси. Сви микросервиси могу да користе различите технологије које одговарају њиховим потребама. Наравно, то не значи да по сваку цену треба мењати технологије али добро је знати да постоји опција уколико за тим постоји потреба. Могућност примене различитих технологија је добра и у погледу задовољства програмера јер им омогућава да усвајају нова знања.

Поред свих наведених предности, микросервисну архитектуру описују и бројни недостаци. Пре свега, изградња оваквих апликација може бити веома ризична уколико се не поседују одговарајућа знања и вештине. Потребни су стручњаци из ове области који ће знати да идентификују микросервисе и управљају њиховом оркестрацијом. Неопходно је адекватно поделити одговорности и функције система. То је једини начин да систем ради како треба и да донесе жељену пословну вредност. У супротном, уколико се овакви системи праве без одговарајућих техничких способности, највероватније се неће исплатити. Такође, познавање саме архитектуре није доволјно јер су потребна и DevOps знања о контејнерима који су уско повезани са микросервисном архитектуром.[4]

Микросервисне апликације су дистрибуирани системи, што их чини знатно сложенијим у односу на монолитне апликације. Већина проблема ове архитектуре произилази из неконтролисаног раста броја микросервиса. Дистрибуирана природа микросервисних апликација отежава процесе који су заједнички за све сервисе. Активности које се сматрају једноставним у монолитном окружењу код микросервиса могу бити знатно отежане. Ту спадају конфигурисање, сигурност, логовање, мониторинг и слично.

Приликом развоја великих, дистрибуираних система, потребно је суочити се са додатном сложеношћу коју они носе са собом. Пример комплексности микросервисних система, у виду броја микросервиса и њихове међусобне комуникације, коју су достигле неке од највећих светских компанија је приказан на слици 20.



Слика 20. Комплексност микросрвисних система [41]

Како постоји међусобна интеракција великог броја сервиса, неопходно је да сваки од њих буде отпоран на могуће грешке у комуникацији или недоступност друге стране. Као решење ове врсте проблема је могуће применити *circuit breaker* патерн. То је механизам који омогућава ефикасан опоравак система од потенцијалних отказа, дефинисањем алтернативних акција које се извршавају у тим ситуацијама. Када дође до грешке, неопходно је што брже идентификовати где је она тачно настала како би могла да се коригује на време и тиме спрече последице већих размера. Најчешће се то ради анализом апликативних логова појединачних сервиса. Међутим, тај посао је готово немогуће радити ручно у системима сачињеним од превеликог броја микросрвиса преко којих се извршава мноштво дистрибуираних операција. Управо у томе се огледа значај постојања мониторинг система у микросрвисним системима, реализованих путем централизованог апликативног логовања. Они обезбеђују централно место за праћење свих важних метрика у реалном времену, како система у целини, тако и појединачних компонената.[39] Циљ је креирати механизам који ће на основу различитих параметара приметити потенцијални проблем пре него што до њега дође и на време обавестити надлежне за његово спречавање.

3.3. Миграција монолитне у микросрвисну архитектуру

Пре самог процеса миграције, неопходно је имати прецизно дефинисану визију, очекивања и разлоге опредељења за микросрвисну архитектуру. Микросрвиси сами по себи нису циљ већ их треба посматрати као средство за постизање циљева који нису оствариви са постојећом архитектуром.[36] То треба схватити јако озбиљно јер миграција комплетног система може бити веома захтеван подухват који изискује значајне трошкове у виду ангажовања нових програмера или мањка времена за рад на редовним функционалностима.

Када су постављени жељени циљеви, у следећем кораку треба дефинисати јасну стратегију миграције. Пре него што се започне процес декомпоновања монолита, треба постићи висок ниво познавања и разумевања постојећег кода. То ће омогућити да се одреде адекватне границе унутар монолита, на основу којих се дефинишу логичке целине које временом треба одвојити у самосталне микросрвисе. Најчешће коришћене стратегије поделе монолита на микросрвисе су:[42]

1. Декомпозиција по бизнис захтеву или функционалности. На пример, *Payment* сервис који је задужен за операције плаћања.
2. Декомпозиција по доменском објекту. На пример, *User* сервис који је задужен за управљање корисницима.

Неадекватно дефинисање граница између микросрвиса узрокује још већи проблем премештања сложености из самог монолита у везе између сервиса, што је много теже за контролисати.[39] Приликом бирања почетних микросрвиса за миграцију, треба узети у обзир оне који су једноставни за издавање и не превише повезани са остатком система. То ће обезбедити видљиве резултате у раној фази миграције, без улагања превеликих напора. Микросрвисна архитектура није препоручљива за системе код којих није могуће одредити јасне границе између сервиса. Лоше постављене границе могу узроковати високу повезаност између сервиса која доводи до проблема да се измене на једном од њих манифестишу и на све остале који су са њим повезани. То знатно отежава и потенцијална рефакторисања јер је премештање кода између сервиса компликованије него када се све налази на једном месту.[39] У таквим околностима монолитна архитектура представља боље решење.

Од кључног значаја у процесу миграције јесте примена инкременталног приступа. Један инкремент обухвата избор неке од постојећих функционалности, њену имплементацију у форми микросрвиса и тестирање да ли све ради као што је планирано. Анализом појединачних корака се добијају боље повратне информације о успешности спроведених измена и корисне смернице везане за даљи рад. Инкрементални приступ омогућава да се временом у производни рад уводи једна по једна функционалност, док би се у

супротном морала чекати миграција целокупног система. Када се врше мање, инкременталне промене, евентуалне грешке су уочљивије па се њихове последице могу ублажити на време. На основу свега наведеног се може закључити да је започињање миграције комплетног система одједном велика грешка.

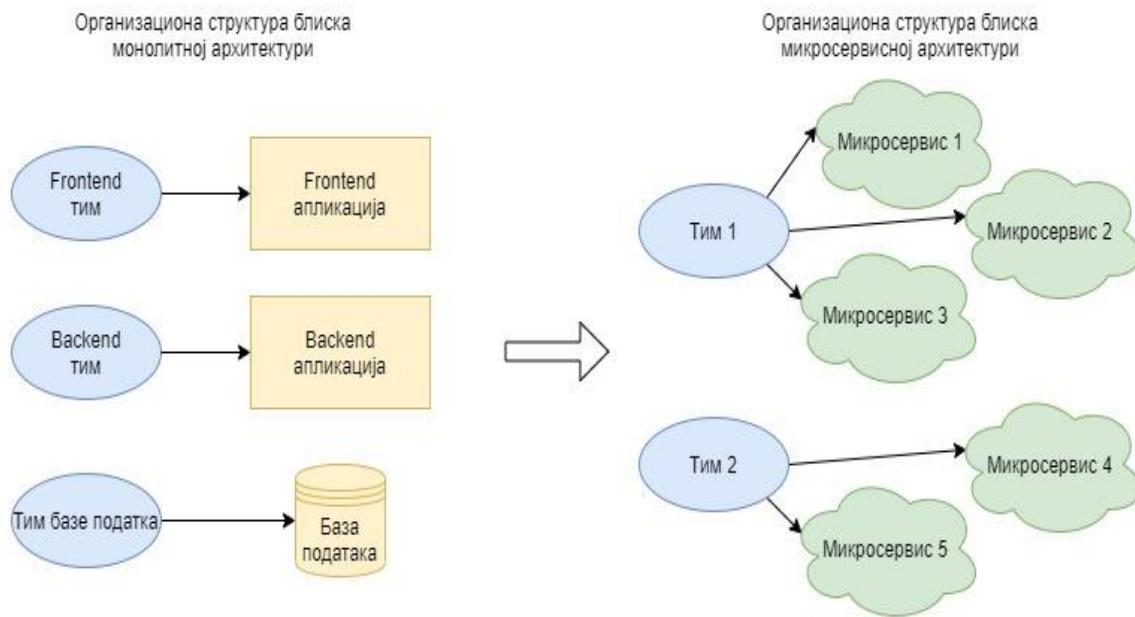
Са имплементационог аспекта, постоје два начина декомпозиције монолитне апликације. Први подразумева једноставно копирање постојећег кода жељене функционалности из монолита у нови микросервис. Овај начин пружа значајну уштеду времена и могућ је само уколико је постојећи код на задовољавајућем нивоу квалитета. Други начин подразумева комплетну реимплементацију постојећег кода и захтева знатно веће количине напора и времена у односу на претходни приступ. Уколико поновна имплементација не изискује исувише много времена и овај приступ представља добро решење јер омогућава да се на мањим инкрементима временом побољшава квалитет система у целини. Код не би требало уклонити из монолитне апликације све док се са сигурношћу не потврди да је миграција успешна. Поступак је приказан на слици 21. На тај начин се добија могућност једноставног враћања на првобитни начин рада уколико се примети да микросервис не функционише као што је планирано. То је врло важно јер процес миграције не би смео да нарушава свакодневно пословање и функционисање система.



Слика 21. Поступна миграција функционалности из монолита у микросервис

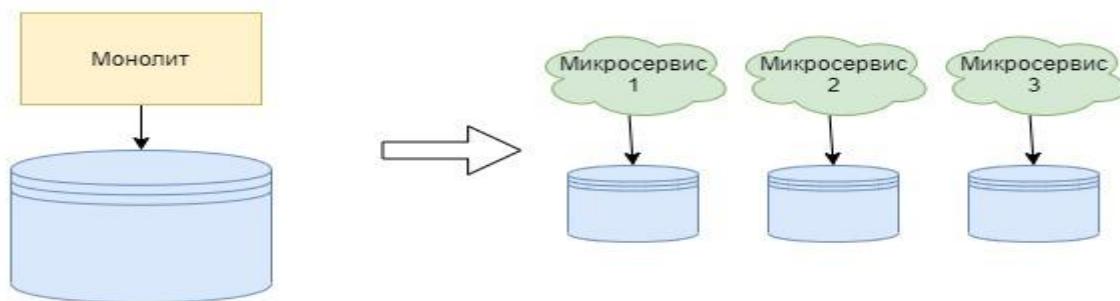
У одређеном тренутку процеса миграције је неохдно извршити прилагођавање организационе структуре новој структури система. У организацијама које примењују монолитну архитектуру је често присутна подела тимова на frontend, backend и тим базе података. Код микросервисне архитектуре се препоручује подела по којој се сваком тиму додељује власништво над одређеном групом сервиса.[36] На тај начин ће бити јасно дефинисана област рада и одговорност сваког појединачног тима. Програмери унутар тима нису више уско усмерени само на одређени слој апликације и скуп технологија повезаних са њим већ су одговорни за све делове микросервиса од почетка до краја. Са тим у вези, потребно је радити и на припреми појединачца у оквиру тимова за нове одговорности које се пред њих стављају, усвајањем знања и вештина које им недостају. Једнако добра опција је и довођење нових стручњака из области која је дефицитарна у оквиру тима уколико за то постоје могућности. На тај начин се, уместо да сви чланови

тима усавршавају нову технологију, ангажује стручњак који већ поседује потребна знања и који временом може упознавати остатак тима са наведеном технологијом. Примери организационих структура блиских монолитној и микросервисној архитектури приказани су на слици 22.



Слика 22. Организационе структуре монолитне и микросервисне архитектуре

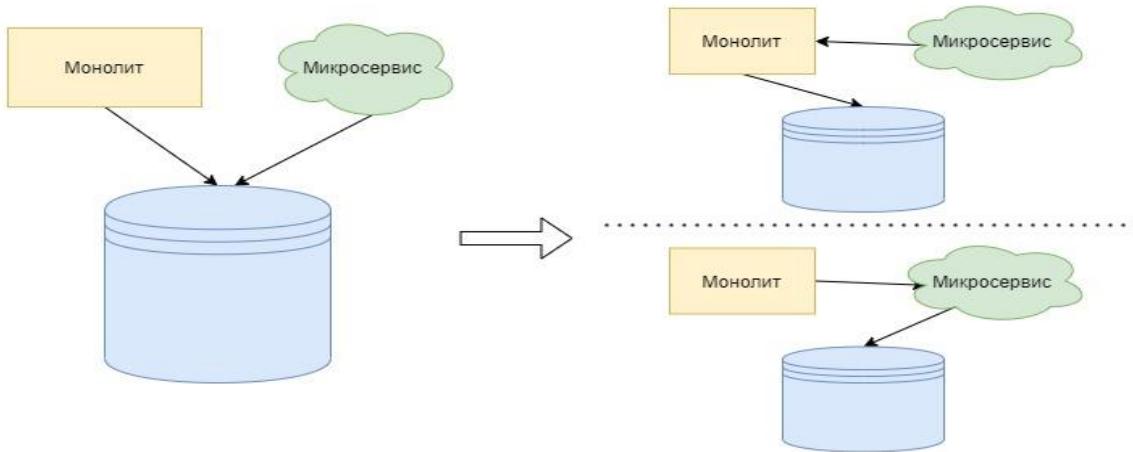
Сматра се да је миграција одређеног дела система комплетна када се код апликације издвоји у посебан микросервис и када су подаци који му припадају издвојени у посебну базу података.[36] Другим речима, неопходно је извршити и декомпозицију монолитне базе података како би процес миграције био потпун. Циљ декомпозиције монолитног система је приказан на слици 23.



Слика 23. Циљ декомпозиције монолитног система

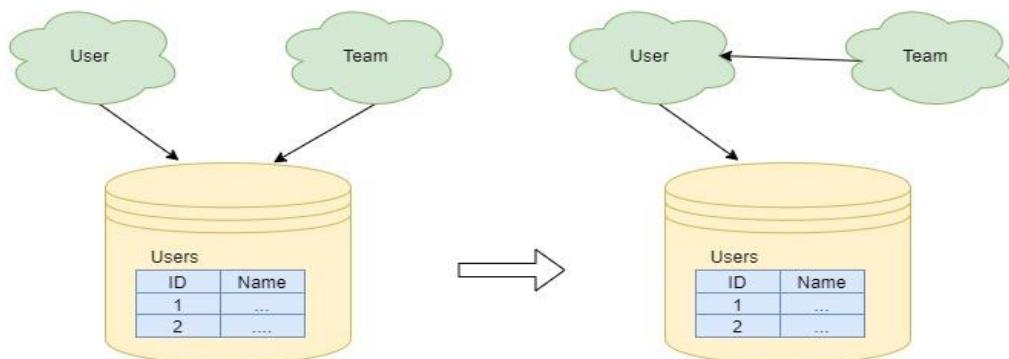
Приликом одређивања припадности података одређеном микросервису треба водити рачуна да се у нови сервис не пренесу подаци који и даље логички припадају монолиту иако их наведени сервис користи за одређене операције. Такве податке треба још увек оставити у монолитном делу система и обезбедити интерфејс путем којег ће им остали сервиси приступати. Са друге стране, ако су подаци успешно одвојени а потребни су и

монолитној апликацији она ће са базе преусмерити позиве ка наведеном микросрвису. Примери дељења података између монолита и микросрвиса су приказани на слици 24.



Слика 24. Дељење података између монолита и микросрвиса

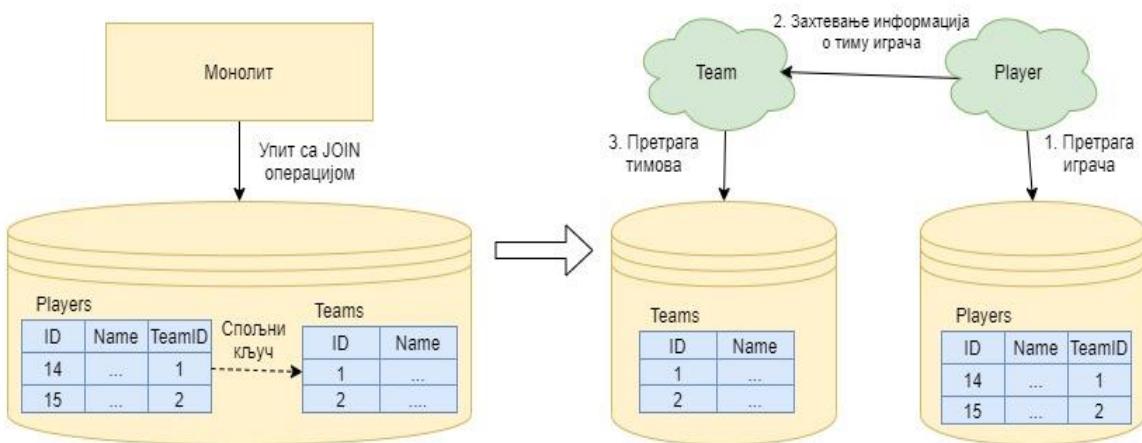
Концепт дељене базе у микросрвисној архитектури носи са собом бројне проблеме. Уколико сви сервиси имају приступ монолитној бази, није могуће вршити контролу приступа одређеним подацима на нивоу сервиса. Такође, не постоји јасно дефинисана одговорност за одређени скуп података, већ је логика за управљање наведеним подацима рас прострањена преко више сервиса. Највећи проблем представља јака повезаност између сервиса који деле базу података. Ако више сервиса користи исту базу а један од њих промени структуру табеле или само назив неке од колона, промена ће се одразити и на све остале. Могуће решење овакве врсте проблема јесте да се директан приступ бази омогући само једном сервису, који ће обезбедити интерфејс за приступ подацима свим заинтересованим клијентима. На тај начин се сакривају имплементациони детаљи и постиже слаба повезаност. На слици 25 је представљен пример где сервиси задужени за управљање корисницима и тимовима приступају заједничкој табели која садржи податке о корисницима. Проблем дељене базе је решен тако што је само User сервису омогућен директан приступ бази, док Team сервис сада преко њега добија потребне податке о корисницима уместо из базе.



Слика 25. Проблем дељене базе података између микросрвиса са предлогом решења

Због свега наведеног, дељену базу треба размотрити само уколико она садржи статичке податке намењене операцији читања. Међутим, некада су околности такве да није могуће декомпоновати базу података. Ако је постојећи систем достигао превисок ниво комплексности, ризик било какве промене је сувише велики јер тада чак и најмања грешка може довести до последица великих размера.

Раздавање базе података носи са собом и одређене недостатке. За извршавање појединих операција је након декомпоновања базе потребан већи број упита а самим тим и више времена. У случају монолитне базе се спајање података из више табела врши једноставном операцијом придрживања (JOIN). Са друге стране, када је база подељена, потребно је извршити упите над више одвојених база података и извршити спајање у самом коду апликације. На слици 26 је представљен пример наведене ситуације. Приказани су сервиси за управљање тимовима и играчима, са припадајућим базама података, где играчи садрже референцу ка тиму коме припадају. Уколико би постојала потреба за свим подацима о одређеном играчу који укључују и детаље о његовом тиму, сервис за управљање играчима би податке о играчу преузео из своје базе док би информације о тиму морао да захтева путем референце од сервиса који је за њих задужен. Након тога се спајање добијених података врши у самом коду. На тај начин су извршена два одвојена позива уместо једног, ефикаснијег упита са операцијом спајања који је могућ у раду са монолитном базом података.



Слика 26. Спајање података у монолитној и микросрввисној архитектури

Такође, са декомпозицијом базе се нарушавају везе између табела повезаних спољним клjuчем уколико се табеле више не налазе у оквиру исте базе. На тај начин се нарушава референцијални интегритет што може довести до неконзистентности података у различитим базама. Другим речима, монолитна база ће спречити брисање реда који је референциран из неке друге табеле, док код микросрвиса то није случај. Тада је потребно осигурати спремност микросрвиса на ситуацију да захтевани податак више не постоји у одвојеној бази података како би се повећала отпорност система.

Раздавање база података знатно отежава и управљање трансакцијама јер нарушава њихова ACID¹ својства. Трансакције које подразумевају операције над различитим базама података не могу имати гарантовану атомност. Саге представљају један од механизама рада са дистрибуираним трансакцијама. Оне подразумевају поделу трансакције на више мањих корака за које су задужени различити сервиси. Уколико дође до грешке у неком од корака, потребно је поништити све претходно потврђене измене. Са тим у вези, сваком кораку треба придружити методу која га поништава и враћа систем у претходно стање.[36]

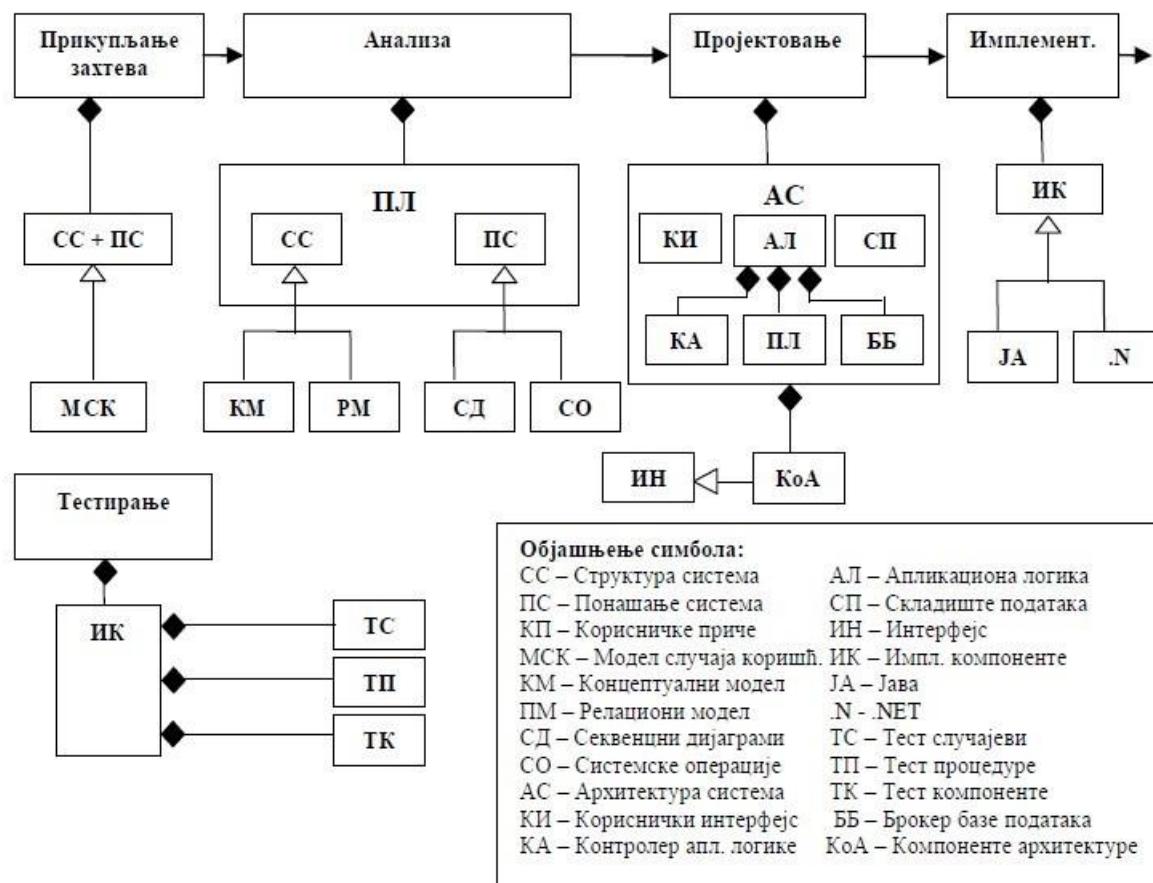
¹ Својства трансакције која обухватају: атомност, конзистентност, изолацију и трајност.

4. Студијски пример

Развој софтверског система, развијеног за потребе овог рада, извршен је применом поједностављене Ларманове методе развоја софтвера. Поједностављена Ларманова метода подразумева да се развој софтверског система састоји од следећих пет фаза:[4]

1. Прикупљање захтева од корисника – Фаза чији је циљ дефинисање захтева које софтверски систем треба да задовољи.
2. Анализа – Фаза која на основу анализе корисничких захтева као резултат даје опис пословне логике система.
3. Пројектовање – Фаза која даје опис тронивојске архитектуре система. Потребно је извршити пројектовање корисничког интерфејса, апликационе логике и складишта података.
4. Имплементација – У овој фази се на основу пројектоване архитектуре система креирају имплементационе компоненте.
5. Тестирање – Фаза у којој се врши тестирање компонената добијених из претходне фазе.

Детаљан приказ фаза Ларманове методе развоја софтвера дат је на слици 27.



Слика 27. Фазе развоја софтверског система по Лармановој методи [4]

4.1. Прикупљање захтева од корисника

У овом одељку је дат детаљан приказ прве фазе поједностављене Ларманове методе развоја софтвера.

4.1.1. Вербални опис модела

Потребно је направити софтверски систем за Football Fantasy игру која ће да води рачуна о корисницима, њиховим тимовима, играчима, клубовима, лигама и мечевима. Приступ систему имају корисник и администратор.

Систем треба да омогући кориснику да се региструје и пријави на систем. Након пријаве, корисник може да креира свој тим избором жељених играча у оквиру ограниченог буџета, при чему тим не сме да садржи више од три играча истог клуба. Затим, корисник може да прегледа мечеве, врши измене над својим тимом, прегледа остварене поене тима и креира нову лигу или се придружи постојећој. Уколико је корисник власник лиге, он је може и обрисати.

Са друге стране, администратор након пријаве може да прегледа, мења или брише играче, као и да прегледа и брише клубове. Администратор може и да врши преглед тимова свих корисника.

4.1.2. Случајеви коришћења

Модел случајева коришћења се састоји од скупа случајева коришћења, актора и веза између њих.[4]

Случај коришћења (СК) представља скуп сценарија тј. скуп жељених коришћења система од стране актора. СК има један главни сценаријо и више алтернативних.[4]

Једну акцију сценарија може извести или актор или систем. Актор изводи следеће акције:[4]

1. Актор припрема улаз за системску операцију (АПУСО)
2. Актор позива систем да изврши системску операцију (АПСО)
3. Актор извршава несистемску операцију (АНСО)

Са друге стране, систем изводи следеће операције: [4]

1. Систем извршава системску операцију (СО)
2. Резултат системске операције (излазни аргументи (ИА)) се прослеђује до актора

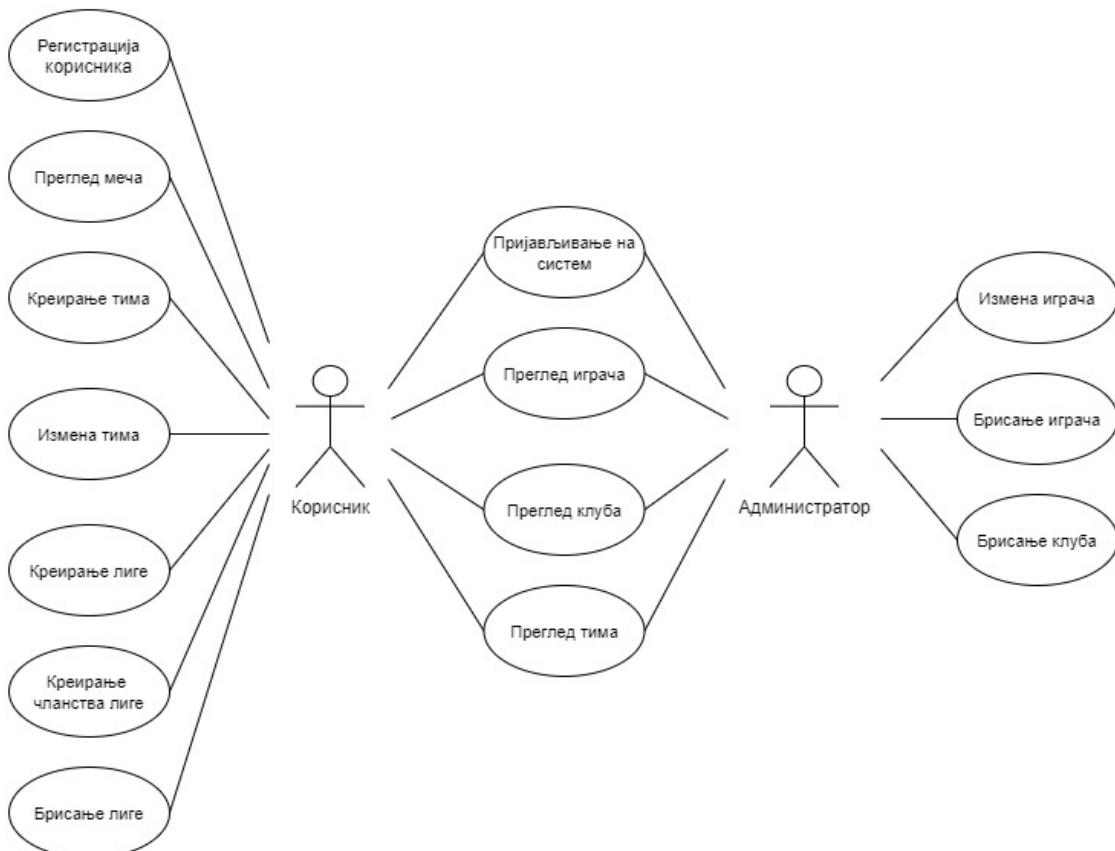
У оквиру овог рада су идентификовани следећи случајеви коришћења и могу се видети на слици 28:

Корисник:

1. Регистрација корисника
2. Пријављивање на систем
3. Преглед меча
4. Креирање тима (Сложен СК)
5. Преглед тима
6. Измена тима
7. Креирање лиге
8. Креирање чланства лиге
9. Брисање лиге
10. Преглед играча
11. Преглед клуба

Администратор:

1. Пријављивање на систем
2. Преглед играча
3. Измена играча
4. Брисање играча
5. Преглед клуба
6. Брисање клуба
7. Преглед тима



Слика 28. Модел случајева коришћења

СК1: Случај коришћења – Регистрација корисника

Назив СК

Регистрација корисника

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за регистрацију. Учитана је листа клубова.

Основни сценарио СК

1. Корисник уноси корисничке податке. (АПУСО)
2. Корисник контролише да ли је коректно унео корисничке податке. (АНСО)
3. Корисник позива систем да запамти корисничке податке. (АПСО)
4. Систем памти податке о кориснику. (СО)
5. Систем приказује кориснику поруку: "Registration completed!". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о кориснику он приказује кориснику поруку: "Error!". (ИА)

СК2: Случај коришћења - Пријављивање на систем

Назив СК

Пријављивање на систем

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за пријављивање.

Основни сценарио СК

1. Актор уноси корисничко име и лозинку. (АПУСО)
2. Актор контролише да ли је коректно унео корисничко име и лозинку. (АНСО)
3. Актор позива систем да провери унете податке. (АПСО)
4. Систем проверава податке о актору. (СО)
5. Систем приказује актору поруку: "Login successful! ". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да нађе актора он приказује актору поруку: "Invalid username or password!". (ИА)

СК3: Случај коришћења – Преглед меча

Назив СК

Преглед меча

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са мечевима. Учитана је листа мечева.

Основни сценарио СК

1. Корисник бира меч којем жели да приступи. (АПУСО)
2. Корисник позива систем да учита податке о одабраном мечу. (АПСО)
3. Систем учитава податке о одабраном мечу. (СО)
4. Систем обавештава корисника о успешном учитавању података о мечу поруком: "Selected match has been loaded!" и приказује податке о одабраном мечу. (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о мечу систем приказује кориснику поруку: "Error!". (ИА)

СК4: Случај коришћења – Креирање тима (Сложен СК)

Назив СК

Креирање тима

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за креирање тима. Учитана је листа играча.

Основни сценарио СК

1. Корисник уноси податке о тиму. (АПУСО)
2. Корисник контролише да ли је коректно унео податке о тиму. (АНСО)
3. Корисник позива систем да запамти податке о тиму. (АПСО)
4. Систем памти податке о тиму. (СО)
5. Систем приказује кориснику поруку: "Team saved!". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о тиму он приказује кориснику поруку: "Error! Team not saved!". (ИА)

СК5: Случај коришћења – Преглед тима

Назив СК

Преглед тима

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са лигама. Учитана је листа лига.

Основни сценарио СК

1. Актор бира тим којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном тиму. (АПСО)
3. Систем учитава податке о одабраном тиму. (СО)
4. Систем обавештава актора о успешном учитавању података о тиму поруком: "Selected team has been loaded!" и приказује податке о одабраном тиму. (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о тиму систем приказује актору поруку: "Error! Selected team can't be loaded!". (ИА)

СК6: Случај коришћења – Измена тима

Назив СК

Измена тима

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са тимом. Учитани су тим и листа играча.

Основни сценарио СК

1. Корисник мења податке о тиму. (АПУСО)
2. Корисник контролише да ли је коректно унео податке о тиму. (АНСО)
3. Корисник позива систем да запамти податке о тиму. (АПСО)
4. Систем памти податке о тиму. (СО)
5. Систем приказује кориснику поруку: "Team saved!". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о тиму он приказује кориснику поруку: "Team not saved!". (ИА)

СК7: Случај коришћења – Креирање лиге

Назив СК

Креирање лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама.

Основни сценарио СК

1. Корисник уноси податке о лиги. (АПУСО)
2. Корисник контролише да ли је коректно унео податке о лиги. (АНСО)
3. Корисник позива систем да запамти податке о лиги. (АПСО)
4. Систем памти податке о лиги. (СО)
5. Систем приказује кориснику поруку: "League created!". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о лиги он приказује кориснику поруку: "Error! League not created!". (ИА)

СК8: Случај коришћења – Креирање чланства лиге

Назив СК

Креирање чланства лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама.

Основни сценарио СК

1. Корисник уноси податке о чланству. (АПУСО)
2. Корисник контролише да ли је коректно унео податке о чланству. (АНСО)
3. Корисник позива систем да запамти податке о чланству. (АПСО)
4. Систем памти податке о чланству. (СО)
5. Систем приказује кориснику поруку: "League joined successfully!". (ИА)

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о чланству он приказује кориснику поруку: "Error! League cannot be joined!". (ИА)

СК9: Случај коришћења – Брисање лиге

Назив СК

Брисање лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама. Учитана је листа лига.

Основни сценарио СК

1. Корисник бира лигу коју жели да обрише. (АПУСО)
2. Корисник позива систем да обрише изабрану лигу. (АПСО)
3. Систем брише лигу. (СО)
4. Систем приказује кориснику поруку: "League deleted!". (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да обрише лигу он приказује кориснику поруку: "Error! League not deleted!". (ИА)

СК10: Случај коришћења – Преглед играча

Назив СК

Преглед играча

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитана је листа играча.

Основни сценарио СК

1. Актор бира играча којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном играчу. (АПСО)
3. Систем учитава податке о одабраном играчу. (СО)
4. Систем обавештава актора о успешном учитавању података о играчу поруком: "Selected player has been loaded!" и приказује податке о одабраном играчу. (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о играчу систем приказује актору поруку: "Error! Selected player cannot be loaded!". (ИА)

СК11: Случај коришћења – Измена играча

Назив СК

Измена играча

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитане су листе играча и клубова.

Основни сценарио СК

1. Администратор бира играча којем жели да приступи. (АПУСО)
2. Администратор позива систем да учита податке о одабраном играчу. (АПСО)
3. Систем учитава податке о одабраном играчу. (СО)
4. Систем обавештава администратора о успешном учитавању података о играчу поруком: "Selected player has been loaded!" и приказује податке о одабраном играчу. (ИА)
5. Администратор мења податке о играчу. (АПУСО)
6. Администратор контролише да ли је коректно унео податке о играчу. (АНСО)
7. Администратор позива систем да запамти податке о играчу. (АПСО)
8. Систем памти податке о играчу. (СО)
9. Систем приказује администратору поруку: "Player is successfully updated!". (ИА)

Алтернативна сценарија

4.1. Уколико систем не може да учита податке о играчу систем приказује администратору поруку: "Error! Selected player cannot be loaded!". Прекида се извршење сценарија. (ИА)

9.1. Уколико систем не може да запамти податке о играчу он приказује администратору поруку: "Error! Player is not updated!". (ИА)

СК12: Случај коришћења – Брисање играча

Назив СК

Брисање играча

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитане су листе играча и клубова.

Основни сценарио СК

1. Администратор бира играча којег жели да обрише. (АПУСО)
2. Администратор позива систем да учита податке о одабраном играчу. (АПСО)
3. Систем учитава податке о одабраном играчу. (СО)
4. Систем обавештава администратора о успешном учитавању података о играчу поруком: "Selected player has been loaded!" и приказује податке о одабраном играчу. (ИА)
5. Администратор позива систем да обрише изабраног играча. (АПСО)
6. Систем брише играча. (СО)
7. Систем приказује администратору поруку: "Player is successfully deleted!". (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о играчу систем приказује администратору поруку: "Error! Selected player cannot be loaded!". Прекида се извршење сценарија. (ИА)
- 7.1. Уколико систем не може да запамти податке о играчу он приказује администратору поруку: "Error! Player is not deleted!". (ИА)

СК13: Случај коришћења – Преглед клуба

Назив СК

Преглед клуба

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са клубовима. Учитана је листа клубова.

Основни сценарио СК

1. Актор бира клуб којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном клубу. (АПСО)
3. Систем учитава податке о одабраном клубу. (СО)
4. Систем обавештава актора о успешном учитавању података о клубу поруком: "Selected club has been loaded!" и приказује податке о одабраном клубу. (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о клубу систем приказује актору поруку: "Error! Selected club cannot be loaded!". (ИА)

СК14: Случај коришћења – Брисање клуба

Назив СК

Брисање клуба

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са клубовима. Учитана је листа клубова.

Основни сценарио СК

1. Администратор бира клуб који жели да обрише. (АПУСО)
2. Администратор позива систем да обрише изабрани клуб. (АПСО)
3. Систем брише клуб. (СО)
4. Систем приказује администратору поруку: "Club deleted successfully!". (ИА)

Алтернативна сценарија

- 4.1. Уколико систем не може да обрише клуб он приказује администратору поруку: "Error! Club can't be deleted!". (ИА)

4.2. Анализа

Анализа је друга по реду фаза Ларманове методе развоја софтвера у којој ће бити дат опис пословне логике система тј. логичке структуре и понашања система. Логичка структура ће бити приказана путем концептуалног и релационог модела; док ће понашање система бити приказано помоћу секвенцијалних дијаграма и системских операција.[4]

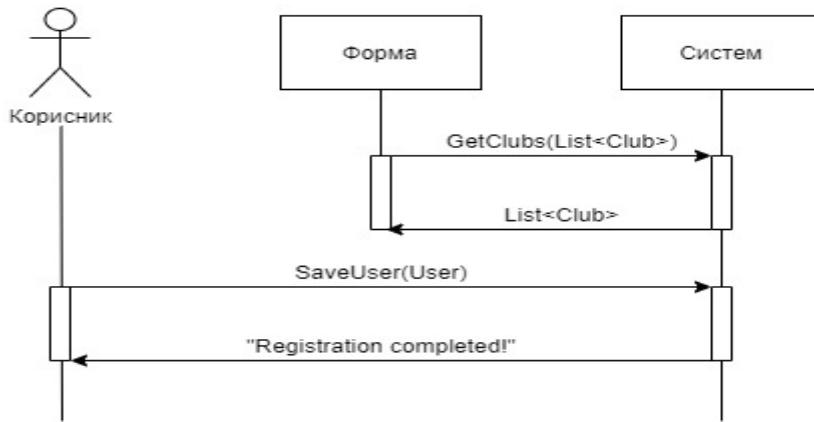
4.2.1. Понашање софтверског система – Системски дијаграм секвенци

Системски дијаграм секвенци за издвојени сценарио СК приказује догађаје у одређеном редоследу, који успостављају интеракцију између актора и софтверског система. Ови дијаграми се праве за сваки сценарио СК и то само за АПСО и ИА акције сценарија.[4]

ДС1: Дијаграм секвенци случаја коришћења – Регистрација корисника

Основни сценарио СК

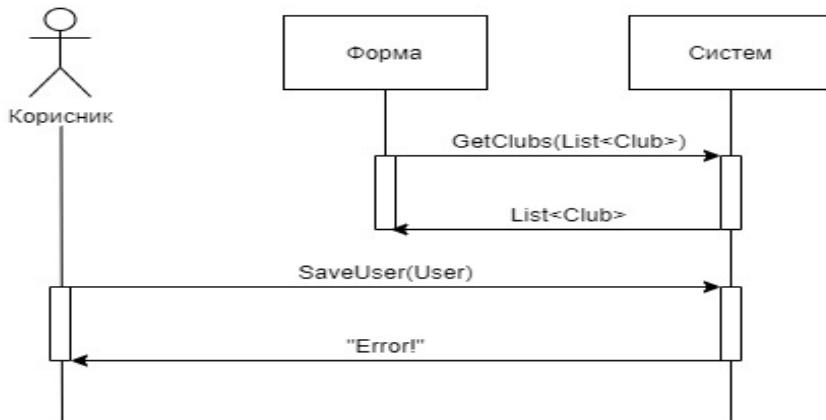
1. **Форма** позива систем да учита листу клубова. (АПСО)
2. **Систем** враћа листу клубова. (ИА)
3. **Корисник** позива **систем** да запамти корисничке податке. (АПСО)
4. **Систем** приказује **кориснику** поруку: "Registration completed!". (ИА)



Дијаграм 1. Дијаграм секвенци СК - Регистрација корисника

Алтернативна сценарија

- 4.1. Уколико **систем** не може да запамти податке о **кориснику** он приказује **кориснику** поруку: "Error!". (ИА)



Дијаграм 2. Дијаграм секвенци СК - Регистрација корисника, алтернативни сценарио 4.1.

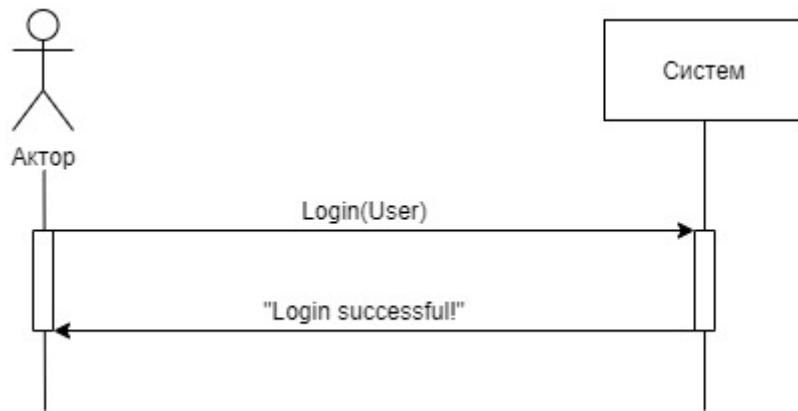
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetClubs(List<Club>);
2. *signal* SaveUser(User);

ДС2: Дијаграм секвенци случаја коришћења - Пријављивање на систем

Основни сценарио СК

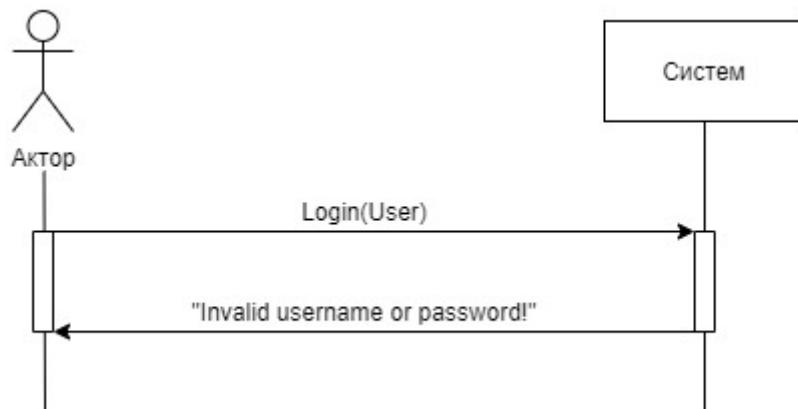
1. **Актор** позива **систем** да провери унете податке. (АПСО)
2. **Систем** приказује **актору** поруку: "Login successful!". (ИА)



Дијаграм 3. Дијаграм секвенци СК - Пријављивање на систем

Алтернативна сценарија

- 2.1. Уколико **систем** не може да нађе **актора** он приказује **актору** поруку: "Invalid username or password!". (ИА)



Дијаграм 4. Дијаграм секвенци СК - Пријављивање на систем, алтернативни сценарио 2.1.

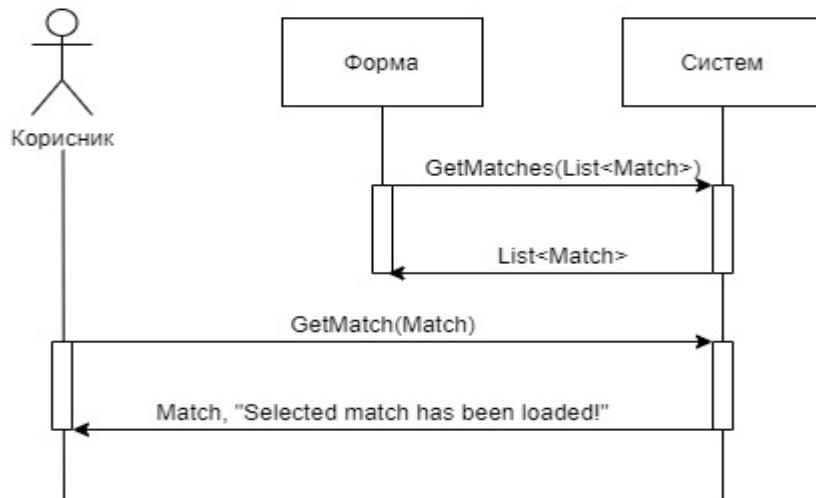
Са наведених секвенцних дијаграма уочава се једна системска операција коју треба пројектовати:

1. `signal Login(User);`

ДСЗ: Дијаграм секвенци случаја коришћења – Преглед меча

Основни сценарио СК

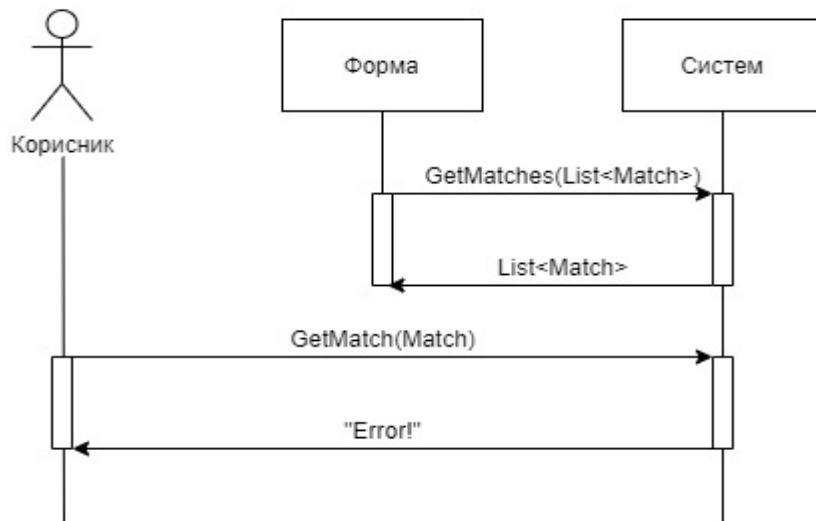
1. **Форма** позива систем да учита листу мечева. (АПСО)
2. **Систем** враћа листу мечева. (ИА)
3. **Корисник** позива **систем** да учита податке о одабраном **мечу**. (АПСО)
4. **Систем** обавештава **корисника** о успешном учитавању података о **мечу** поруком: "Selected match has been loaded!" и приказује податке о одабраном **мечу**. (ИА)



Дијаграм 5. Дијаграм секвенци СК - Преглед меча

Алтернативна сценарија

- 4.1. Уколико **систем** не може да учита податке о **мечу** **систем** приказује **кориснику** поруку: "Error!". (ИА)



Дијаграм 6. Дијаграм секвенци СК - Преглед меча, алтернативни сценарио 4.1.

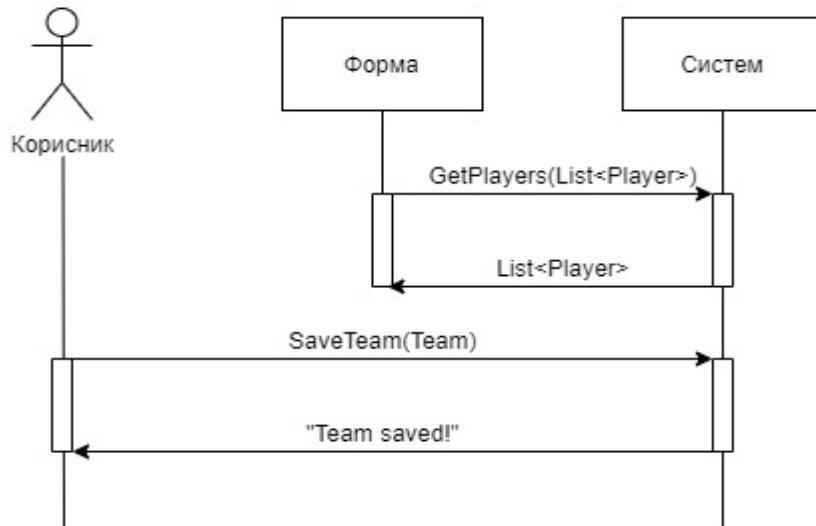
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. `signal GetMatches(List<Match>);`
2. `signal GetMatch(Match);`

ДС4: Дијаграм секвенци случаја коришћења – Креирање тима (Сложен СК)

Основни сценарио СК

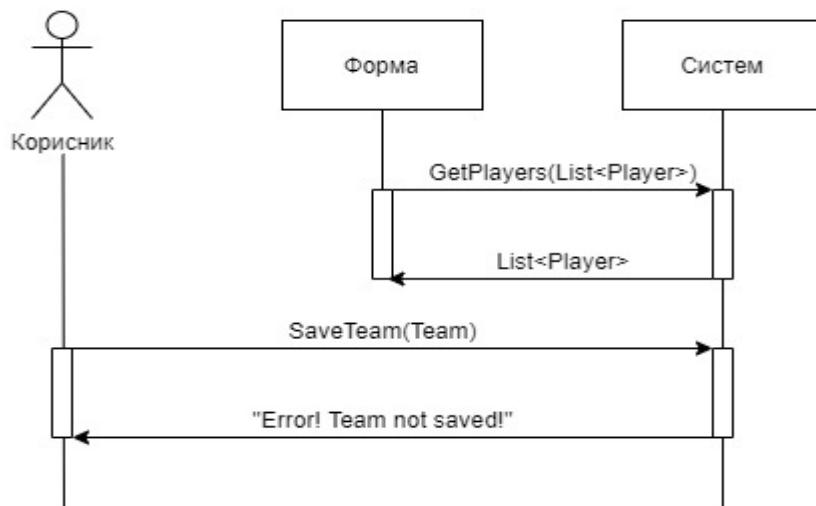
1. **Форма** позива систем да учита листу играча. (АПСО)
2. **Систем** враћа листу играча. (ИА)
3. **Корисник** позива **систем** да запамти податке о **тиму**. (АПСО)
4. **Систем** приказује **кориснику** поруку: "Team saved!". (ИА)



Дијаграм 7. Дијаграм секвенци СК - Креирање тима (Сложен СК)

Алтернативна сценарија

- 4.1. Уколико **систем** не може да запамти податке о **тиму** он приказује **кориснику** поруку: "Error! Team not saved!". (ИА)



Дијаграм 8. Дијаграм секвенци СК - Креирање тима (Сложен СК), алтернативни сценарио 4.1.

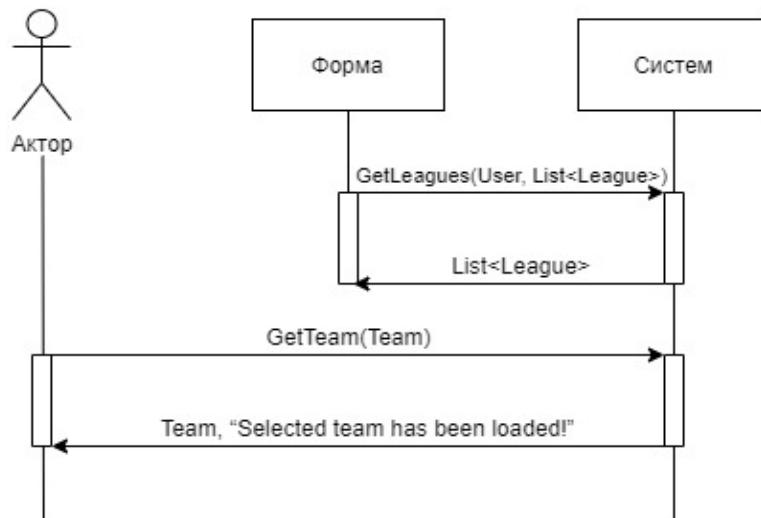
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetPlayers(List<Player>);
2. *signal* SaveTeam(Team);

ДС5: Дијаграм секвенци случаја коришћења – Преглед тима

Основни сценарио СК

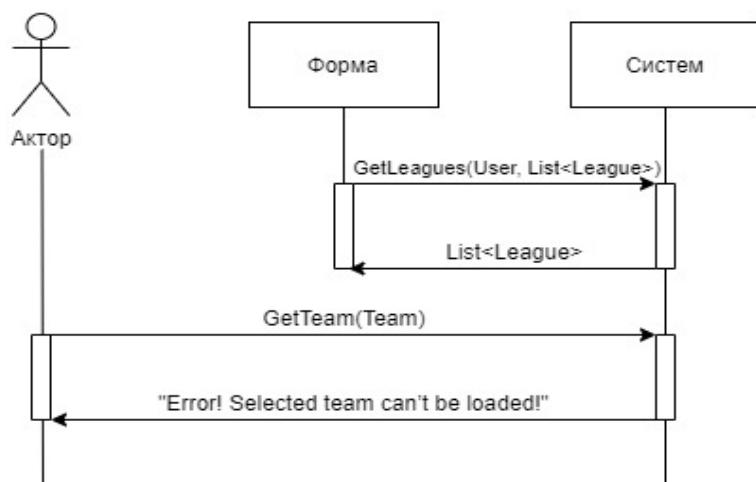
1. **Форма позива** систем да учита листу лига. (АПСО)
2. **Систем враћа** листу лига. (ИА)
3. **Актор позива** систем да учита податке о одабраном **тиму**. (АПСО)
4. **Систем обавештава** **актора** о успешном учитавању података о **тиму** поруком: "Selected team has been loaded!" и приказује податке о одабраном **тиму**. (ИА)



Дијаграм 9. Дијаграм секвенци СК - Преглед тима

Алтернативна сценарија

- 4.1. Уколико **систем** не може да учита податке о **тиму** **систем** приказује **актору** поруку: "Error! Selected team can't be loaded!". (ИА)



Дијаграм 10. Дијаграм секвенци СК - Преглед тима, алтернативни сценарио 4.1.

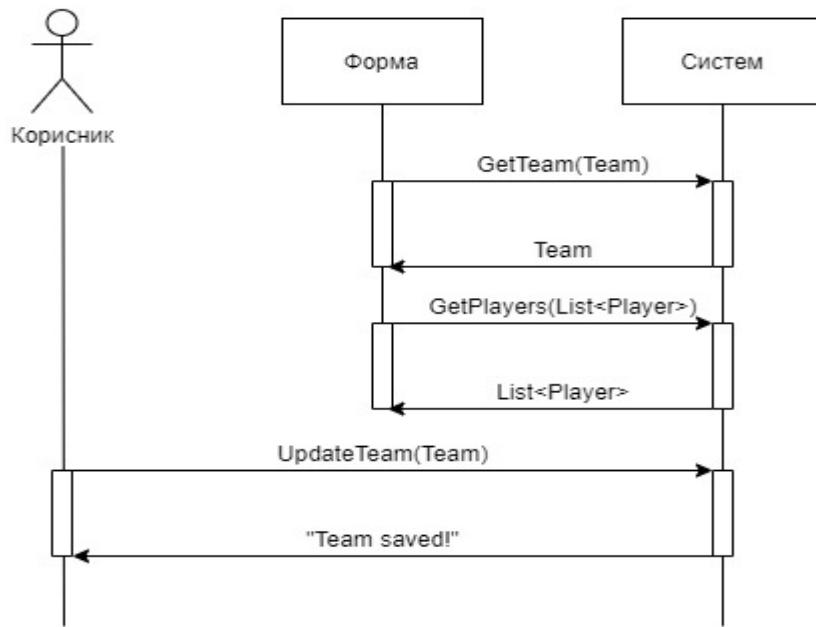
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetLeagues(User, List<League>);
2. *signal* GetTeam(Team);

ДС6: Дијаграм секвенци случаја коришћења – Измена тима

Основни сценарио СК

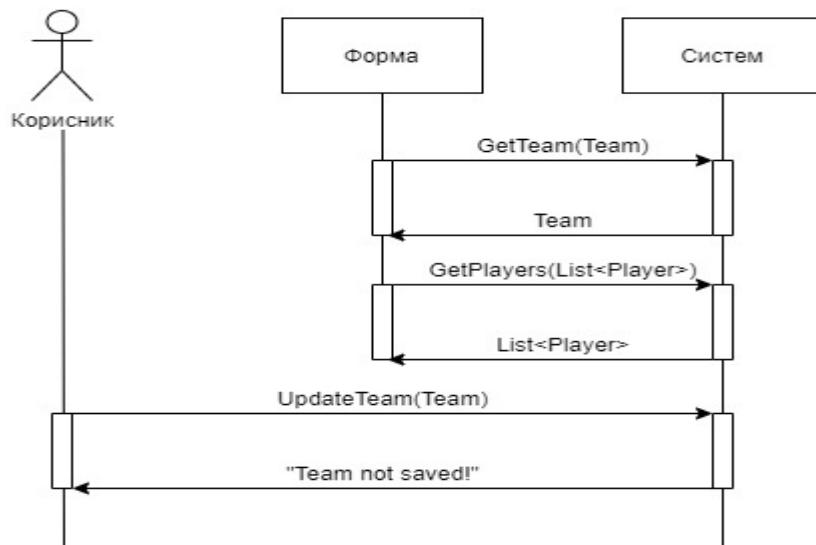
1. **Форма позива** систем да учита тим. (АПСО)
2. **Систем враћа** тим. (ИА)
3. **Форма позива** систем да учита листу играча. (АПСО)
4. **Систем враћа** листу играча. (ИА)
5. **Корисник позива** систем да запамти податке о **тиму**. (АПСО)
6. **Систем приказује** кориснику поруку: "Team saved!". (ИА)



Дијаграм 11. Дијаграм секвенци СК - Измена тима

Алтернативна сценарија

- 6.1. Уколико **систем** не може да запамти податке о **тиму** он приказује **кориснику** поруку: "Team not saved!". (ИА)



Дијаграм 12. Дијаграм секвенци СК - Измена тима, алтернативни сценарио 6.1.

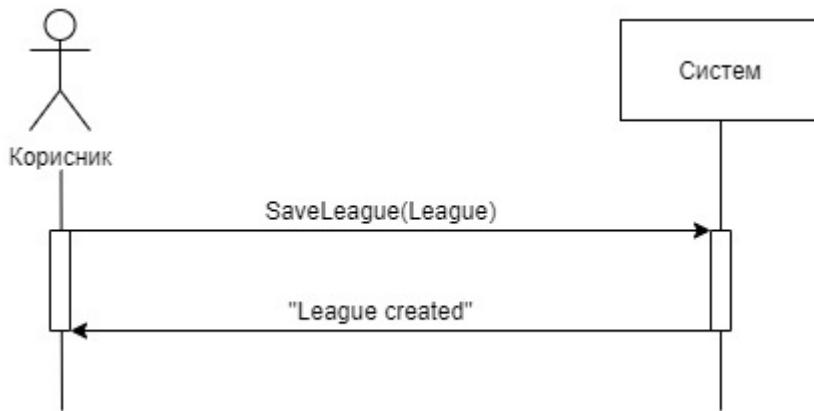
Са наведених секвенцних дијаграма уочавају се три системске операције које треба пројектовати:

1. *signal* GetTeam(Team);
2. *signal* GetPlayers(List<Player>);
3. *signal* UpdateTeam(Team);

ДС7: Дијаграм секвенци случаја коришћења – Креирање лиге

Основни сценарио СК

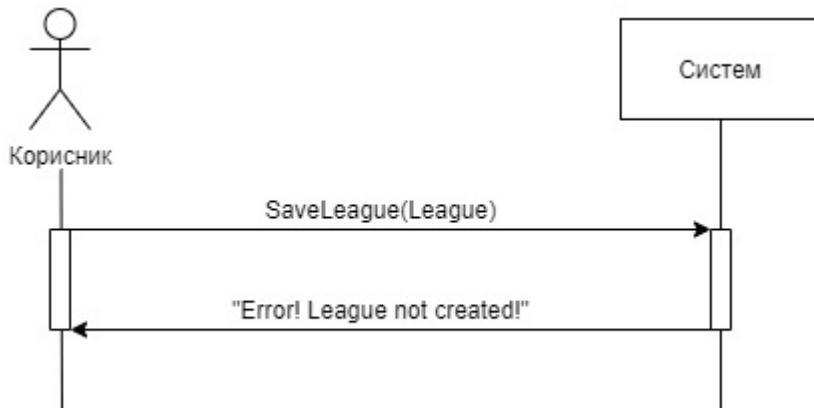
1. Корисник позива систем да запамти податке о лиги. (АПСО)
2. Систем приказује кориснику поруку: "League created!". (ИА)



Дијаграм 13. Дијаграм секвенци СК - Креирање лиге

Алтернативна сценарија

- 2.1. Уколико систем не може да запамти податке о лиги он приказује кориснику поруку: "Error! League not created!". (ИА)



Дијаграм 14. Дијаграм секвенци СК - Креирање лиге, алтернативни сценарио 2.1.

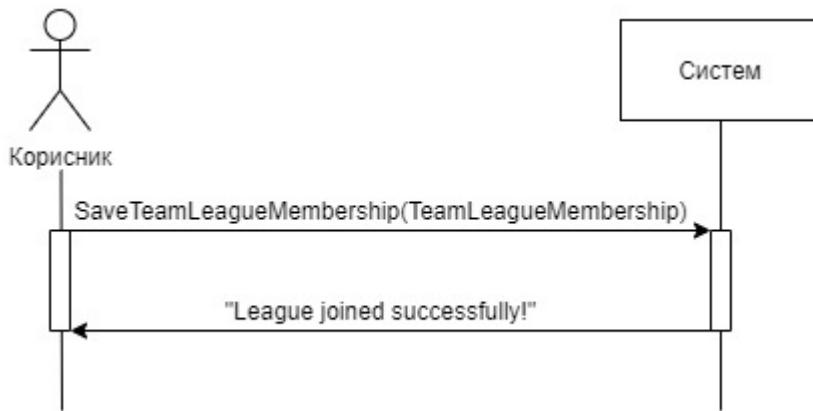
Са наведених секвенцних дијаграма уочава се једна системска операција коју треба пројектовати:

1. signal SaveLeague(League);

ДС8: Дијаграм секвенци случаја коришћења – Креирање чланства лиге

Основни сценарио СК

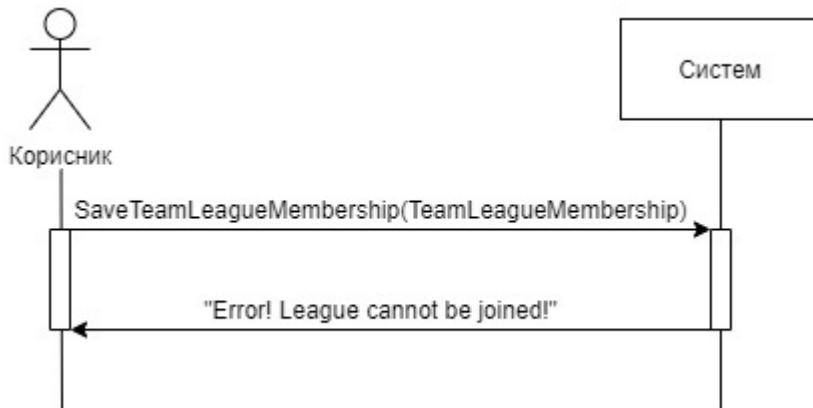
1. Корисник позива систем да запамти податке о чланству. (АПСО)
2. Систем приказује кориснику поруку: "League joined successfully!". (ИА)



Дијаграм 15. Дијаграм секвенци СК - Креирање чланства лиге

Алтернативна сценарија

- 2.1. Уколико систем не може да запамти податке о чланству он приказује кориснику поруку: "Error! League cannot be joined!". (ИА)



Дијаграм 16. Дијаграм секвенци СК - Креирање чланства лиге, алтернативни сценарио 2.1.

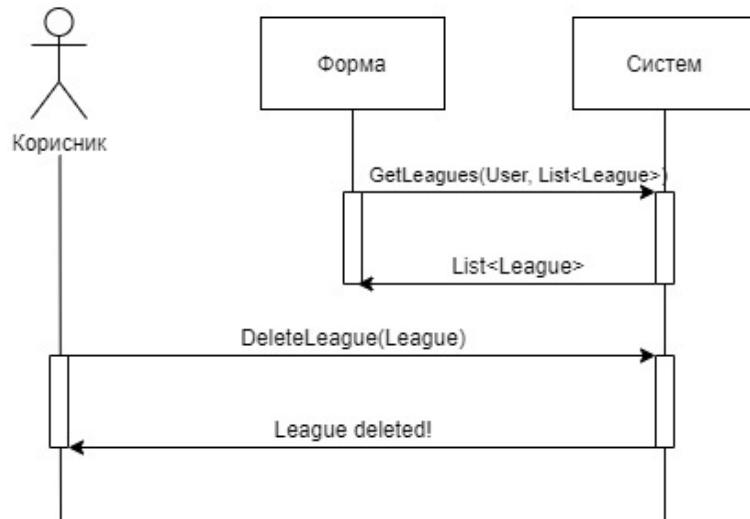
Са наведених секвенцних дијаграма уочава се једна системска операција коју треба пројектовати:

1. *signal SaveTeamLeagueMembership(TeamLeagueMembership);*

ДС9: Дијаграм секвенци случаја коришћења – Брисање лиге

Основни сценарио СК

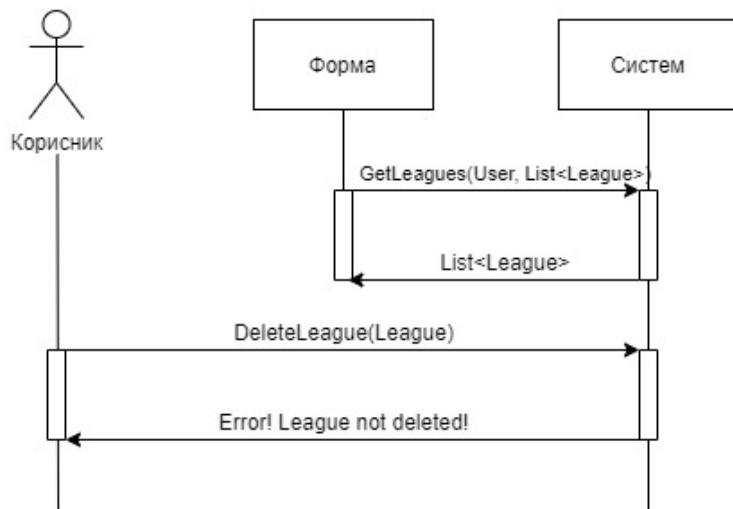
1. **Форма** позива систем да учита листу лига. (АПСО)
2. **Систем** враћа листу лига. (ИА)
3. **Корисник** позива **систем** да обрише изабрану **лигу**. (АПСО)
4. **Систем** приказује **кориснику** поруку: "League deleted!". (ИА)



Дијаграм 17. Дијаграм секвенци СК - Брисање лиге

Алтернативна сценарија

- 4.1. Уколико **систем** не може да обрише **лигу** он приказује **кориснику** поруку: "Error! League not deleted!". (ИА)



Дијаграм 18. Дијаграм секвенци СК - Брисање лиге, алтернативни сценарио 4.1.

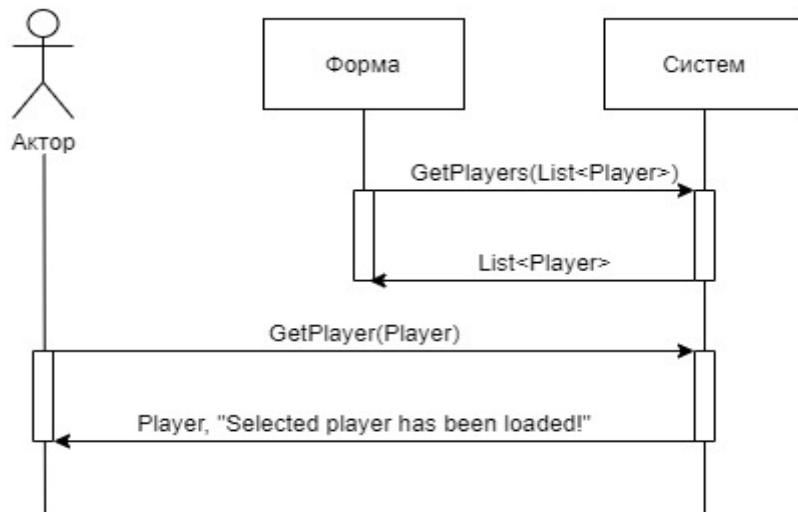
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetLeagues(User, List<League>);
2. *signal* DeleteLeague(League);

ДС10: Дијаграм секвенци случаја коришћења – Преглед играча

Основни сценарио СК

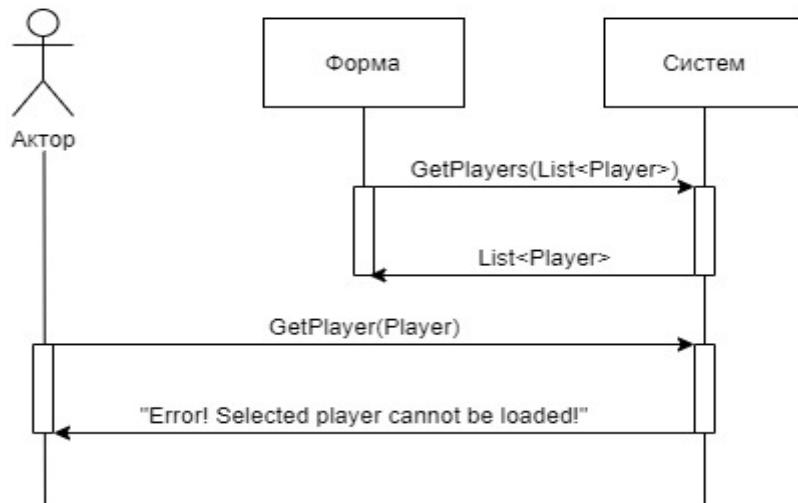
1. **Форма** позива систем да учита листу играча. (АПСО)
2. **Систем** враћа листу играча. (ИА)
3. **Актор** позива **систем** да учита податке о одабраном **играчу**. (АПСО)
4. **Систем** обавештава **актора** о успешном учитавању података о **играчу** поруком: "Selected player has been loaded!" и приказује податке о одабраном **играчу**. (ИА)



Дијаграм 19. Дијаграм секвенци СК - Преглед играча

Алтернативна сценарија

- 4.1. Уколико **систем** не може да учита податке о **играчу** **систем** приказује **актору** поруку: "Error! Selected player cannot be loaded!". (ИА)



Дијаграм 20. Дијаграм секвенци СК - Преглед играча, алтернативни сценарио 4.1.

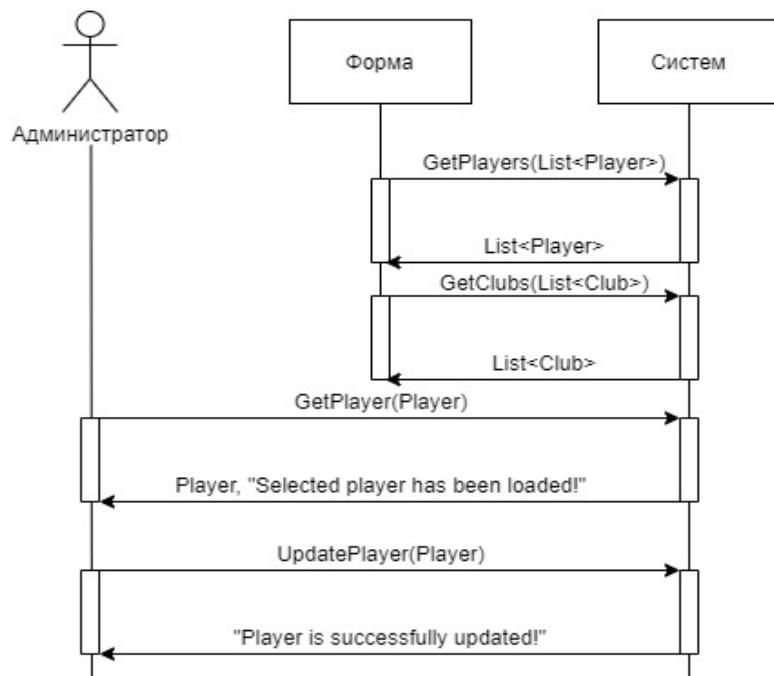
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal GetPlayers(List<Player>);*
2. *signal GetPlayer(Player);*

ДС11: Дијаграм секвенци случаја коришћења – Измена играча

Основни сценарио СК

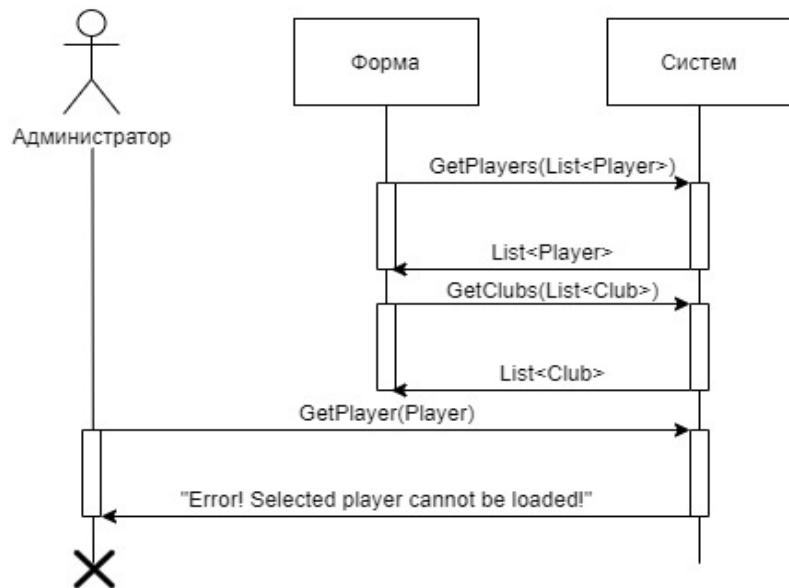
1. **Форма позива** систем да учита листу играча. (АПСО)
2. **Систем враћа** листу играча. (ИА)
3. **Форма позива** систем да учита листу клубова. (АПСО)
4. **Систем враћа** листу клубова. (ИА)
5. **Администратор позива** систем да учита податке о одабраном **играчу**. (АПСО)
6. **Систем обавештава администратора** о успешном учитавању података о **играчу** поруком: "Selected player has been loaded!" и приказује податке о одабраном **играчу**. (ИА)
7. **Администратор позива** систем да запамти податке о **играчу**. (АПСО)
8. **Систем приказује администратору** поруку: "Player is successfully updated!". (ИА)



Дијаграм 21. Дијаграм секвенци СК - Измена играча

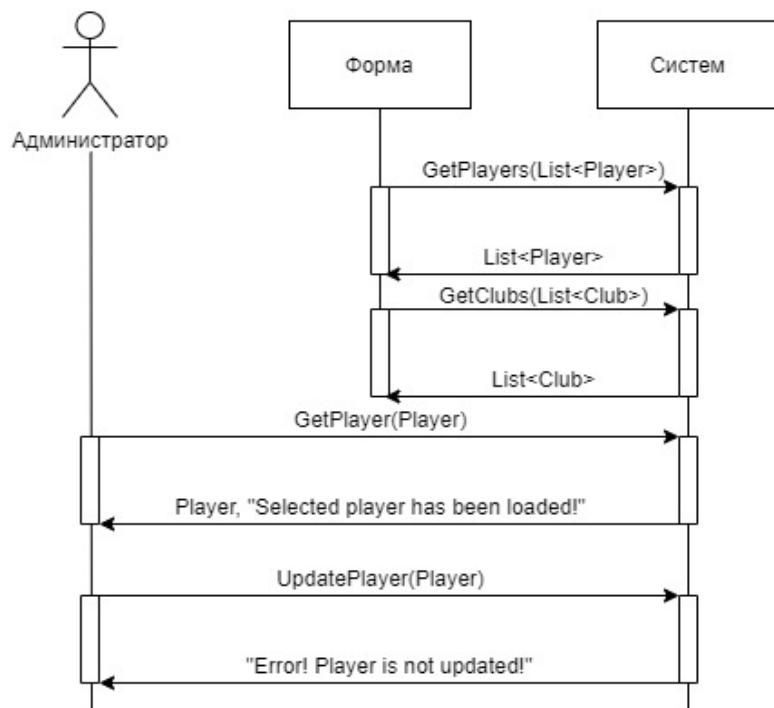
Алтернативна сценарија

- 6.1. Уколико **систем** не може да учита податке о **играчу** **систем** приказује **администратору** поруку: "Error! Selected player cannot be loaded!". Прекида се извршење сценарија. (ИА)



Дијаграм 22. Дијаграм секвенци СК - Измена играча, алтернативни сценарио 6.1.

8.1. Уколико **систем** не може да запамти податке о играчу он приказује **администратору** поруку: "Error! Player is not updated!". (ИА)



Дијаграм 23. Дијаграм секвенци СК - Измена играча, алтернативни сценарио 8.1.

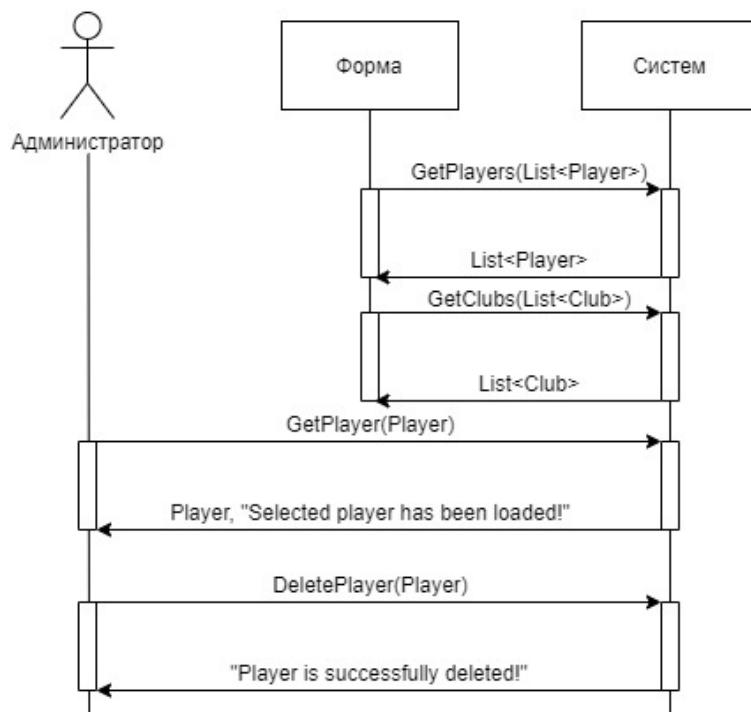
Са наведених секвенцних дијаграма уочавају се четири системске операције које треба пројектовати:

1. `signal GetPlayers(List<Player>);`
2. `signal GetClubs(List<Club>);`
3. `signal GetPlayer(Player);`
4. `signal UpdatePlayer(Player);`

ДС12: Дијаграм секвенци случаја коришћења – Брисање играча

Основни сценарио СК

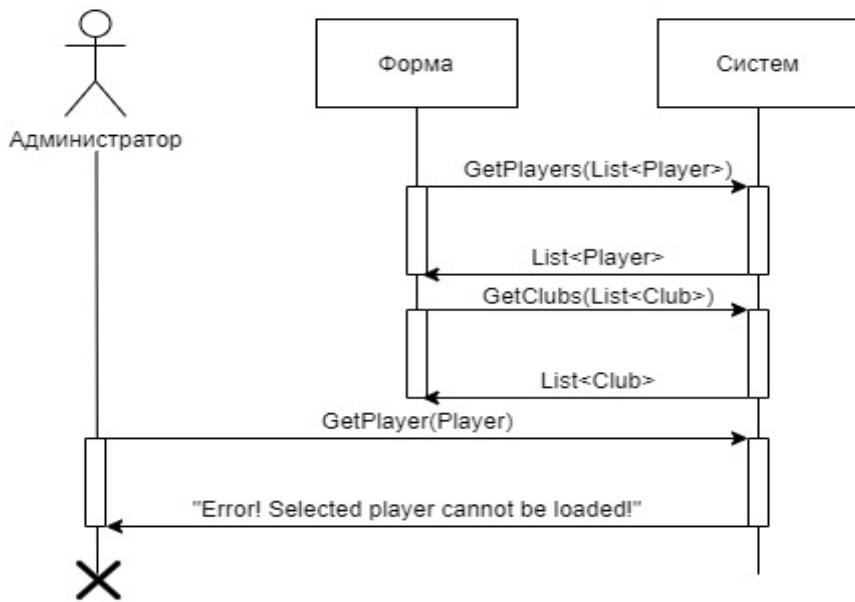
1. **Форма позива** систем да учита листу играча. (АПСО)
2. **Систем враћа** листу играча. (ИА)
3. **Форма позива** систем да учита листу клубова. (АПСО)
4. **Систем враћа** листу клубова. (ИА)
5. **Администратор позива** систем да учита податке о одабраном **играчу**. (АПСО)
6. **Систем обавештава администратора** о успешном учитавању података о **играчу** поруком: "Selected player has been loaded!" и приказује податке о одабраном **играчу**. (ИА)
7. **Администратор позива** систем да обрише изабраног **играча**. (АПСО)
8. **Систем приказује администратору** поруку: "Player is successfully deleted!". (ИА)



Дијаграм 24. Дијаграм секвенци СК - Брисање играча

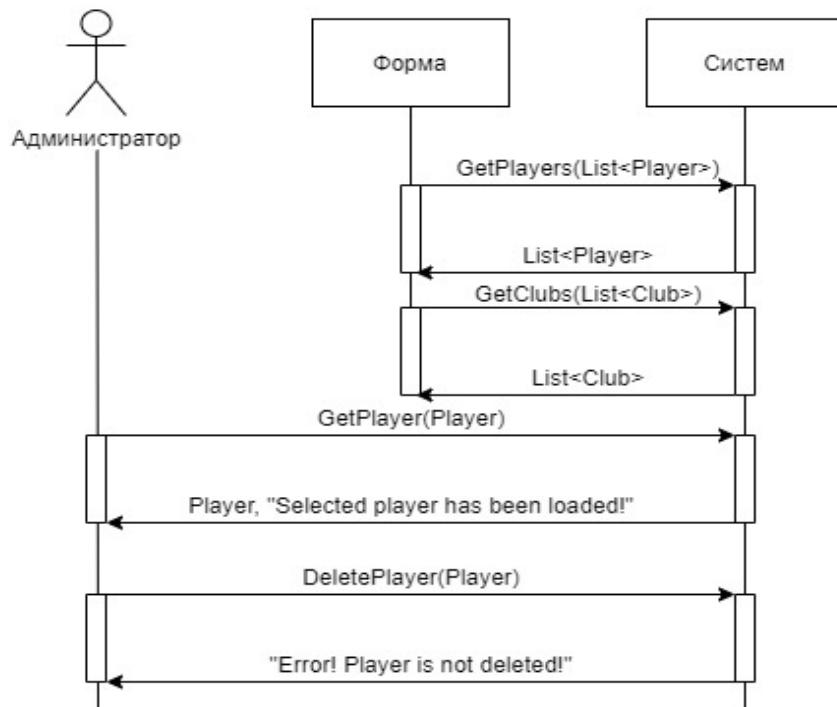
Алтернативна сценарија

- 6.1. Уколико **систем** не може да учита податке о **играчу** **систем** приказује **администратору** поруку: "Error! Selected player cannot be loaded!". Прекида се извршење сценарија. (ИА)



Дијаграм 25. Дијаграм секвенци СК - Брисање играча, алтернативни сценарио 6.1.

8.1. Уколико **систем** не може да запамти податке о играчу он приказује **администратору** поруку: "Error! Player is not deleted!". (ИА)



Дијаграм 26. Дијаграм секвенци СК - Брисање играча, алтернативни сценарио 8.1.

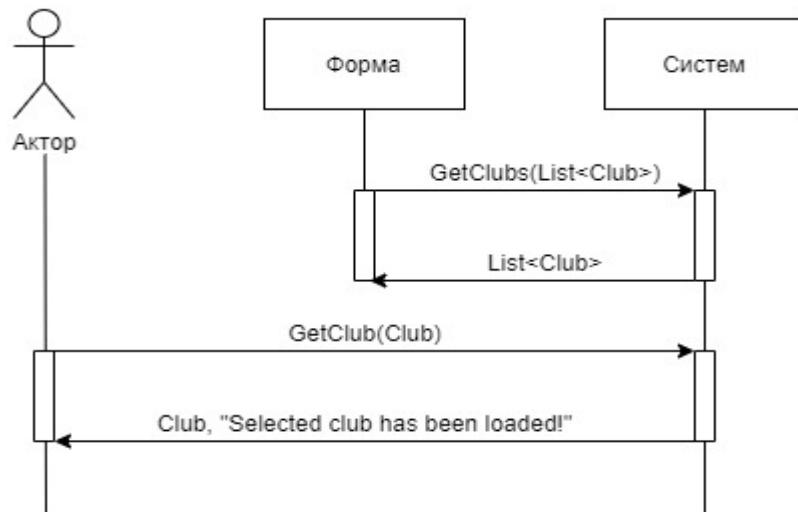
Са наведених секвенцних дијаграма уочавају се четири системске операције које треба пројектовати:

1. *signal GetPlayers(List<Player>);*
2. *signal GetClubs(List<Club>);*
3. *signal GetPlayer(Player);*
4. *signal DeletePlayer(Player);*

ДС13: Дијаграм секвенци случаја коришћења – Преглед клуба

Основни сценарио СК

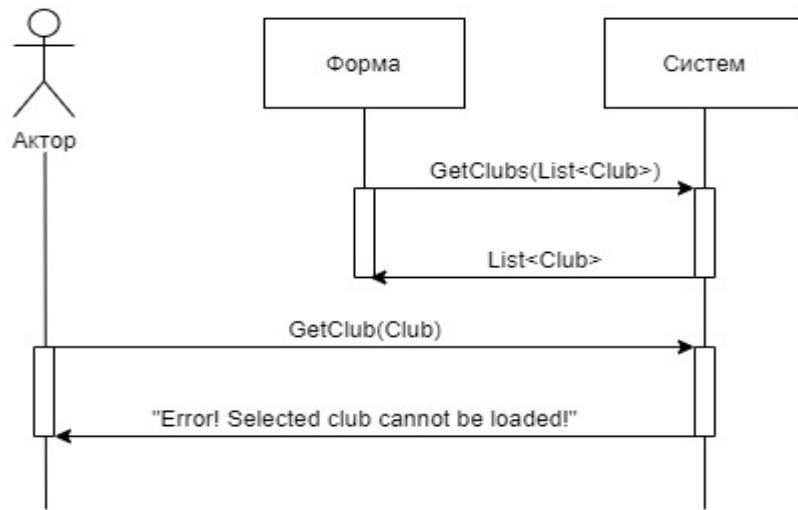
1. **Форма** позива систем да учита листу клубова. (АПСО)
2. **Систем** враћа листу клубова. (ИА)
3. **Актор** позива **систем** да учита податке о одабраном **клубу**. (АПСО)
4. **Систем** обавештава **актора** о успешном учитавању података о **клубу** поруком: "Selected club has been loaded!" и приказује податке о одабраном **клубу**. (ИА)



Дијаграм 27. Дијаграм секвенци СК - Преглед клуба

Алтернативна сценарија

- 4.1. Уколико **систем** не може да учита податке о **клубу** **систем** приказује **актору** поруку: "Error! Selected club cannot be loaded!". (ИА)



Дијаграм 28. Дијаграм секвенци СК - Преглед клуба, алтернативни сценарио 4.1.

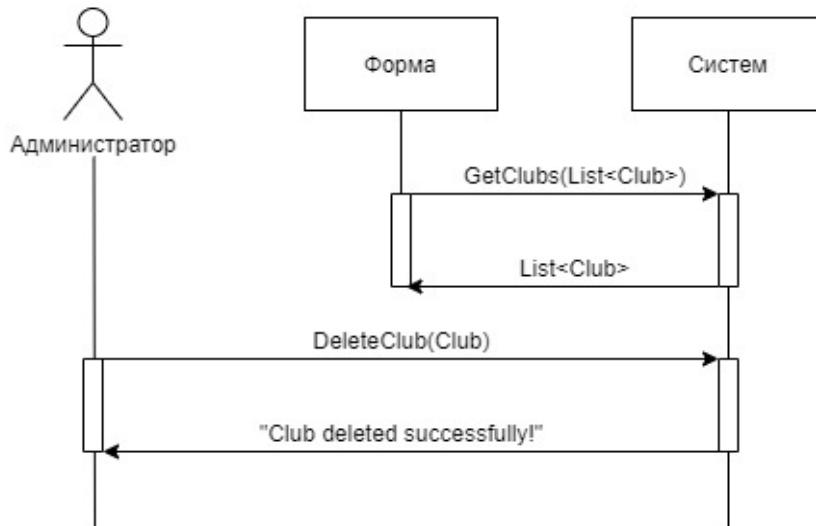
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetClubs(List<Club>);
2. *signal* GetClub(Club);

ДС14: Дијаграм секвенци случаја коришћења – Брисање клуба

Основни сценарио СК

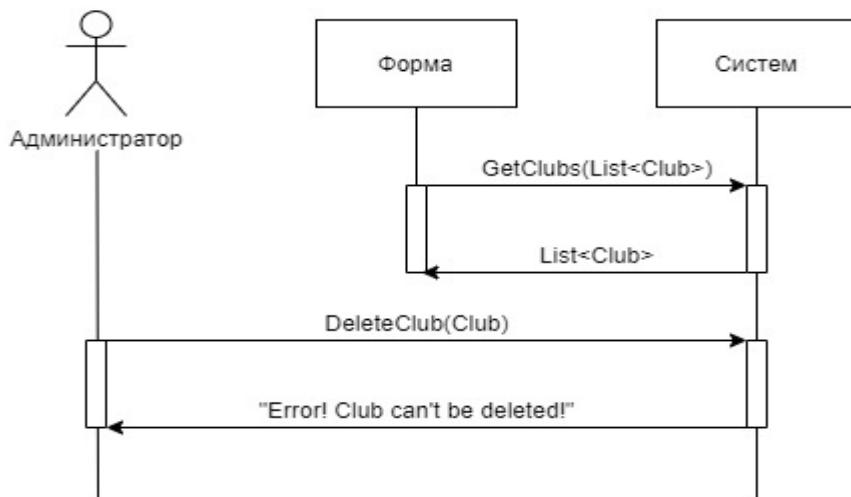
1. **Форма** позива систем да учита листу клубова. (АПСО)
2. **Систем** враћа листу клубова. (ИА)
3. **Администратор** позива **систем** да обрише изабрани **клуб**. (АПСО)
4. **Систем** приказује **администратору** поруку: "Club deleted successfully!". (ИА)



Дијаграм 29. Дијаграм секвенци СК - Брисање клуба

Алтернативна сценарија

- 4.1. Уколико **систем** не може да обрише **клуб** он приказује **администратору** поруку: "Error! Club can't be deleted!". (ИА)



Дијаграм 30. Дијаграм секвенци СК - Брисање клуба, алтернативни сценарио 4.1.

Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* GetClubs(List<Club>);
2. *signal* DeleteClub(Club);

Како резултат анализе сценарија добијено је укупно 18 системских операција које треба пројектовати:

1. *signal* GetClubs(List<Club>);
2. *signal* SaveUser(User);
3. *signal* Login(User);
4. *signal* GetMatches(List<Match>);
5. *signal* GetMatch(Match);
6. *signal* GetPlayers(List<Player>);
7. *signal* SaveTeam(Team);
8. *signal* GetLeagues(User, List<League>);
9. *signal* GetTeam(Team);
10. *signal* UpdateTeam(Team);
11. *signal* SaveLeague(League);
12. *signal* SaveTeamLeagueMembership(TeamLeagueMembership);
13. *signal* DeleteLeague(League);
14. *signal* GetPlayer(Player);
15. *signal* UpdatePlayer(Player);
16. *signal* DeletePlayer(Player);
17. *signal* GetClub(Club);
18. *signal* DeleteClub(Club);

4.2.2. Понашање софтверског система – Дефинисање уговора о системским операцијама

Уговори се праве за сваку уочену системску операцију и они описују њено понашање. Сваки уговор се састоји из следећих елемената: [4]

- **Операција** – име операције и њени улазни и излазни елементи
- **Веза са СК** – имена случајева којима се позива СО
- **Предуслов** – предуслови који морају бити задовољени пре извршења СО
- **Постуслови** – постуслови који морају бити задовољени у систему након извршења СО

Уговор УГ1: GetClubs

Операција: **GetClubs**(List<Club>): signal;

Веза са СК: CK1, CK11, CK12, CK13, CK14

Предуслови: /

Постуслови: /

Уговор УГ2: SaveUser

Операција: **SaveUser**(User): signal;

Веза са СК: CK1

Предуслови: Вредносна и структурна ограничења над објектом User морају бити задовољена.

Постуслови: Креiran је нови корисник.

Уговор УГ3: Login

Операција: **Login**(User): signal;

Веза са СК: CK2

Предуслови: /

Постуслови: /

Уговор УГ4: GetMatches

Операција: **GetMatches**(List<Match>): signal;

Веза са СК: CK3

Предуслови: /

Постуслови: /

Уговор УГ5: GetMatch

Операција: **GetMatch**(Match): signal;

Веза са СК: CK3

Предуслови: /

Постуслови: /

Уговор УГ6: GetPlayers

Операција: **GetPlayers**(List<Player>): signal;

Веза са СК: CK4, CK6, CK10, CK11, CK12

Предуслови: /

Постуслови: /

[Уговор УГ7: SaveTeam](#)

Операција: **SaveTeam**(Team): signal;

Веза са СК: CK4

Предуслови: Вредносна и структурна ограничења над објектом Team морају бити задовољена.

Постуслови: Креiran је нови тим.

[Уговор УГ8: GetLeagues](#)

Операција: **GetLeagues**(User, List<League>): signal;

Веза са СК: CK5, CK9

Предуслови: /

Постуслови: /

[Уговор УГ9: GetTeam](#)

Операција: **GetTeam**(Team): signal;

Веза са СК: CK5, CK6

Предуслови: /

Постуслови: /

[Уговор УГ10: UpdateTeam](#)

Операција: **UpdateTeam**(Team): signal;

Веза са СК: CK6

Предуслови: Вредносна и структурна ограничења над објектом Team морају бити задовољена.

Постуслови: Тим је изменењен.

[Уговор УГ11: SaveLeague](#)

Операција: **SaveLeague**(League): signal;

Веза са СК: CK7

Предуслови: Вредносна и структурна ограничења над објектом League морају бити задовољена.

Постуслови: Креирана је нова лига.

[Уговор УГ12: SaveTeamLeagueMembership](#)

Операција: **SaveTeamLeagueMembership**(TeamLeagueMembership): signal;

Веза са СК: CK8

Предуслови: Вредносна и структурна ограничења над објектом

TeamLeagueMembership морају бити задовољена.

Постуслови: Креирано је ново чланство лиге.

[Уговор УГ13: DeleteLeague](#)

Операција: **DeleteLeague**(League): signal;

Веза са СК: CK9

Предуслови: Структурна ограничења над објектом League морају бити задовољена.

Постуслови: Лига је обрисана.

[Уговор УГ14: GetPlayer](#)

Операција: **GetPlayer**(Player): signal;

Веза са СК: CK10, CK11, CK12

Предуслови: /

Постуслови: /

[Уговор УГ15: UpdatePlayer](#)

Операција: **UpdatePlayer**(Player): signal;

Веза са СК: CK11

Предуслови: Вредносна и структурна ограничења над објектом Player морају бити задовољена.

Постуслови: Играч је изменењен.

[Уговор УГ16: DeletePlayer](#)

Операција: **DeletePlayer**(Player): signal;

Веза са СК: CK12

Предуслови: Структурна ограничења над објектом Player морају бити задовољена.

Постуслови: Играч је обрисан.

[Уговор УГ17: GetClub](#)

Операција: **GetClub**(Club): signal;

Веза са СК: CK13

Предуслови: /

Постуслови: /

[Уговор УГ18: DeleteClub](#)

Операција: **DeleteClub**(Club): signal;

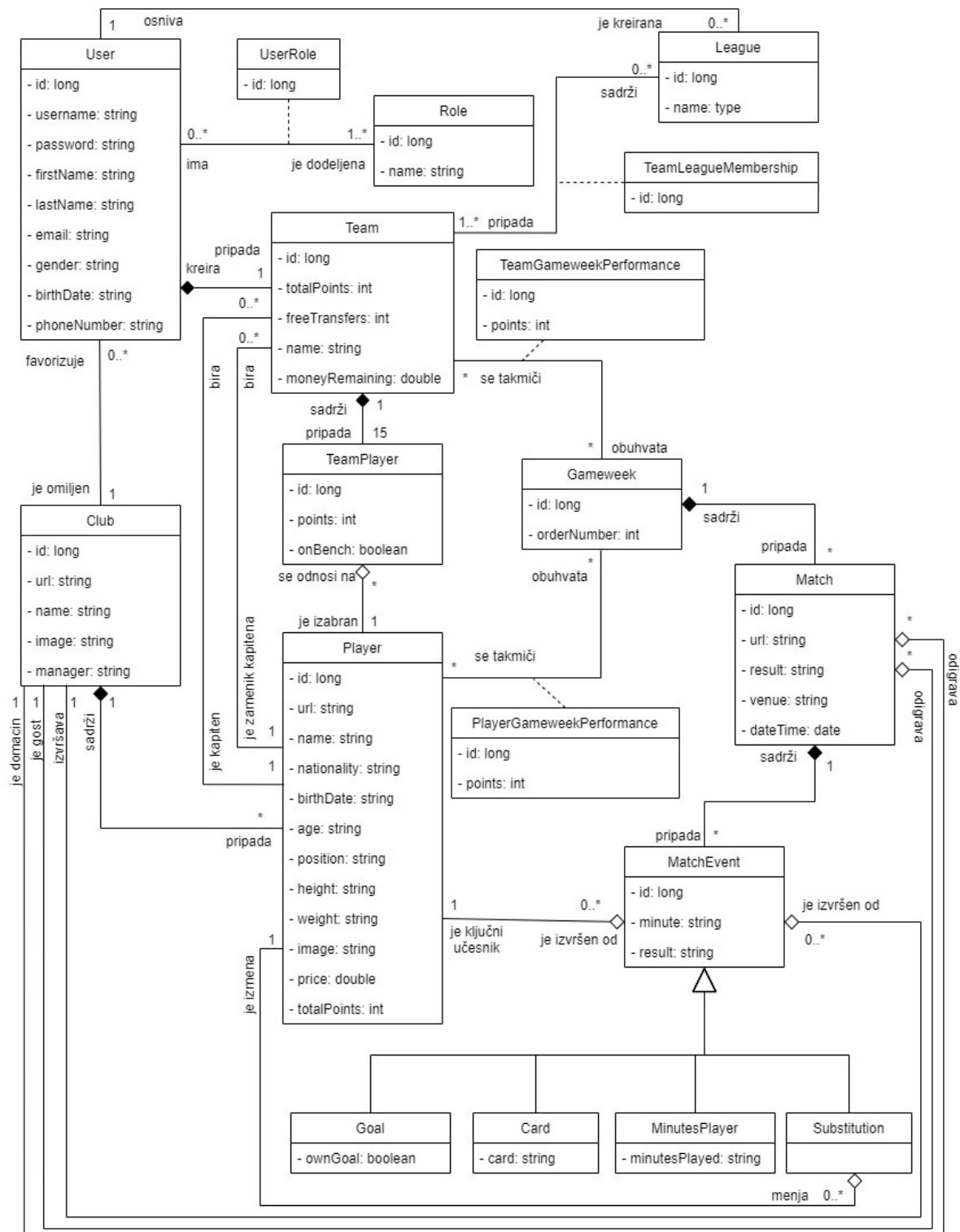
Веза са СК: CK14

Предуслови: Структурна ограничења над објектом Club морају бити задовољена.

Постуслови: Клуб је обрисан.

4.2.3. Структура софтверског система – Концептуални (доменски) модел

Концептуални модел приказује концептуалне класе и асоцијације између њих. Концептуалне класе садрже атрибуте који описују особине класе. Асоцијација представља везу између концептуалних класа, при чему сваки крај асоцијације означава улогу класе која учествује у асоцијацији.[4]



Дијаграм 31. Концептуални модел

4.2.4 Структура софтверског система – Релациони модел

Релациони модел се креира на основу добијеног концептуалног модела.

User(*id*, *username*, *password*, *firstName*, *lastName*, *email*, *gender*, *birthDate*, *phoneNumber*, *favouriteClubId*)

Role(*id*, *name*)

UserRole(*id*, *userId*, *roleId*)

Club(*id*, *url*, *name*, *image*, *manager*)

Player(*id*, *clubId*, *url*, *name*, *nationality*, *birthDate*, *age*, *position*, *height*, *weight*, *image*, *price*, *totalPoints*)

Team(*id*, *userId*, *totalPoints*, *freeTransfers*, *name*, *moneyRemaining*, *captainId*, *viceCaptainId*)

TeamPlayer(*id*, *teamId*, *points*, *onBench*, *playerId*)

League(*id*, *name*, *ownerId*)

TeamLeagueMembership(*id*, *teamId*, *leagueId*)

Gameweek(*id*, *orderNumber*)

TeamGameweekPerformance(*id*, *teamId*, *gameweekId*, *points*)

PlayerGameweekPerformance(*id*, *playerId*, *gameweekId*, *points*)

Match(*id*, *gameweekId*, *url*, *result*, *dateTime*, *venue*, *hostId*, *guestId*)

MatchEvent(*id*, *matchId*, *gameweekId*, *minute*, *result*, *clubId*, *playerId*)

Goal(*matchEventId*, *matchId*, *gameweekId*, *ownGoal*)

Card(*matchEventId*, *matchId*, *gameweekId*, *card*)

MinutesPlayed(*matchEventId*, *matchId*, *gameweekId*, *minutesPlayed*)

Substitution(*matchEventId*, *matchId*, *gameweekId*, *outPlayerId*)

Табела User		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	id	long	not null			INSERT RESTRICTED Club
	username	String	not null			UPDATE RESTRICTED Club
	password	String	not null			CASCADES Team, League, UserRole
	firstName	String	not null			
	lastName	String	not null			
	email	String	not null			
	gender	String	not null			
	birthDate	String	not null			
	phoneNumber	String	not null			
	favouriteClubId	long	not null			

Табела 1. Табела User

Табела Role		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	id	long	not null			INSERT / UPDATE CASCADES UserRole
	name	String	not null			DELETE RESTRICTED UserRole

Табела 2. Табела Role

Табела UserRole		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	id	long	not null			INSERT RESTRICTED User, Role
	userId	long	not null			UPDATE RESTRICTED User, Role
	roleId	long	not null			DELETE /

Табела 3. Табела UserRole

Табела Club		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	id	long	not null			INSERT / UPDATE CASCADES User, Player, MatchEvent, Match
	url	String	not null			
	name	String	not null			
	image	String	not null			
	manager	String	not null			DELETE RESTRICTED User, Match, MatchEvent CASCADES Player

Табела 4. Табела Club

Табела Player		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Club UPDATE RESTRICTED Club CASCADES TeamPlayer, Team, MatchEvent, Substitution, PlayerGameweek Performance DELETE RESTRICTED TeamPlayer, Team, MatchEvent, Substitution, PlayerGameweek Performance
	id	long	not null			
	clubId	String	not null			
	url	String	not null			
	name	String	not null			
	nationality	String	not null			
	birthdate	String	not null			
	age	String	not null			
	position	String	not null			
	height	String	not null			
	weight	String	not null			
	image	String	not null			
	price	double	not null and > 0			
	totalPoints	int	not null		totalPoints= SUM(Player GameweekPerformance. points)	

Табела 5. Табела Player

Табела Team		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED User, Player UPDATE RESTRICTED User, Player CASCADES TeamPlayer, TeamLeagueMembership, TeamGameweek Performance DELETE CASCADES TeamPlayer RESTRICTED TeamLeagueMembership, TeamGameweek Performance
	id	long	not null			
	userId	long	not null			
	totalPoints	int	not null		totalPoints= SUM(Team GameweekPerformance. points)	
	freeTransfers	int	not null and >= 0			
	name	String	not null			
	moneyRemaining	double	not null and > 0			
	captainId	long	not null			
	viceCaptainId	long	not null			

Табела 6. Табела Team

Табела TeamPlayer		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Team, Player UPDATE RESTRICTED Team, Player DELETE /
	id	long	not null			
	teamId	long	not null			
	points	int	not null			
	onBench	boolean	not null			
	playerId	long	not null			

Табела 7. Табела TeamPlayer

Табела League		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED User UPDATE RESTRICTED User CASCADES TeamLeagueMembership DELETE RESTRICTED TeamLeagueMembership
	id	long	not null			
	name	String	not null			
	ownerId	long	not null			

Табела 8. Табела League

Табела TeamLeagueMembership		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Team, League UPDATE RESTRICTED Team, League DELETE /
	id	long	not null			
	teamId	long	not null			
	leagueId	long	not null			

Табела 9. Табела TeamLeagueMembership

Табела Gameweek		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT / UPDATE CASCADES Match, TeamGameweek Performance, PlayerGameweekPerformance DELETE CASCADES Match RESTRICTED TeamGameweek Performance, PlayerGameweekPerformance
	id	long	not null			
	orderNumber	int	not null and > 0			

Табела 10. Табела Gameweek

Табела TeamGameweekPerformance		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Team, Gameweek UPDATE RESTRICTED Team, Gameweek DELETE /
	id	long	not null			
	teamId	long	not null			
	gameweekId	long	not null			
	points	int	not null		points=SUM(TeamPlayer.points)	

Табела 11. Табела TeamGameweekPerformance

Табела PlayerGameweekPerformance		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Player, Gameweek
	id	long	not null			UPDATE RESTRICTED Player, Gameweek
	playerId	long	not null			DELETE /
	gameweekId	long	not null			
	points	int	not null			

Табела 12. PlayerGameweekPerformance

Табела Match		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Gameweek,Club
	id	long	not null			UPDATE RESTRICTED Gameweek,Club
	gameweekId	long	not null			CASCADES MatchEvent
	url	String	not null			DELETE CASCADES MatchEvent
	result	String	not null			
	dateTime	date	not null			
	venue	String	not null			
	hostId	long	not null			
	guestId	long	not null			

Табела 13. Табела Match

Табела MatchEvent		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED Match, Club, Player
	id	long	not null			UPDATE RESTRICTED Match, Club, Player
	matchId	long	not null			DELETE /
	gameweekId	long	not null			
	minute	String	not null			
	result	String	not null			
	clubId	long	not null			
	playerId	long	not null			

Табела 14. Табела MatchEvent

Табела Goal		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT RESTRICTED MatchEvent
	matchEventId	long	not null			UPDATE RESTRICTED MatchEvent
	matchId	long	not null			DELETE RESTRICTED MatchEvent
	gameweekId	long	not null			
	ownGoal	boolean	not null			

Табела 15. Табела Goal

Табела Card		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	matchEventId	long	not null			INSERT RESTRICTED MatchEvent
	matchId	long	not null			UPDATE RESTRICTED MatchEvent
	gameweekId	long	not null			DELETE RESTRICTED MatchEvent
	card	String	not null			

Табела 16. Табела Card

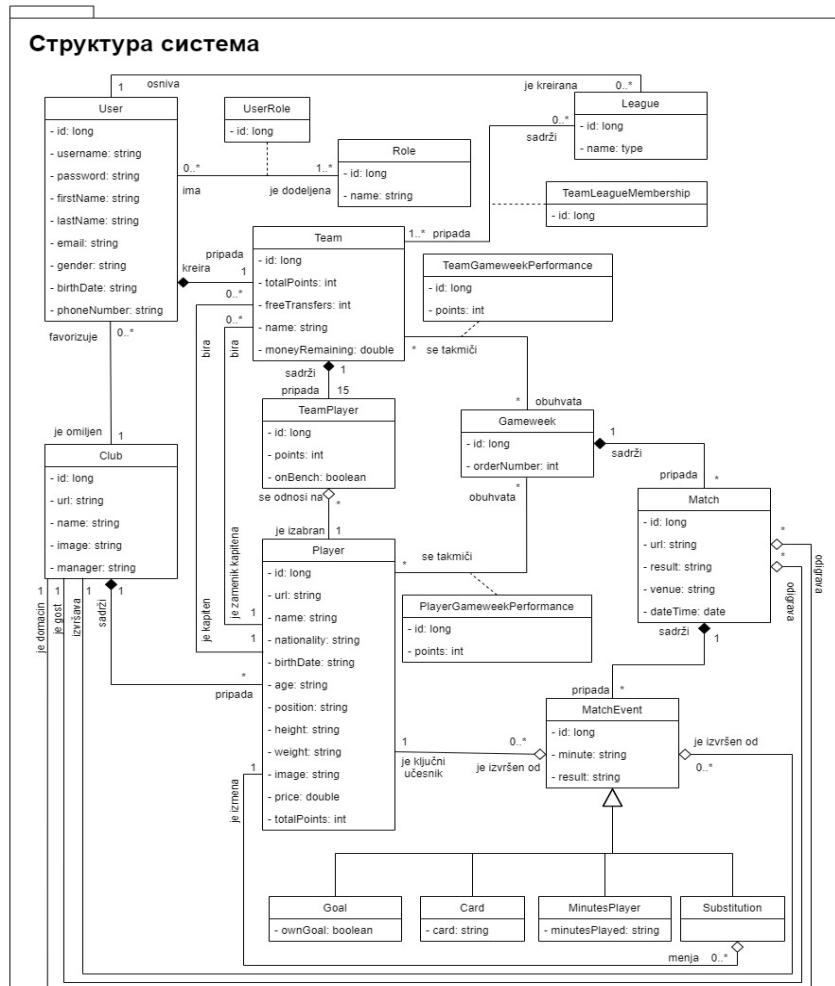
Табела MinutesPlayed		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	matchEventId	long	not null			INSERT RESTRICTED MatchEvent
	matchId	long	not null			UPDATE RESTRICTED MatchEvent
	gameweekId	long	not null			DELETE RESTRICTED MatchEvent
	minutesPlayed	int	not null and >= 0			

Табела 17. Табела MinutesPlayed

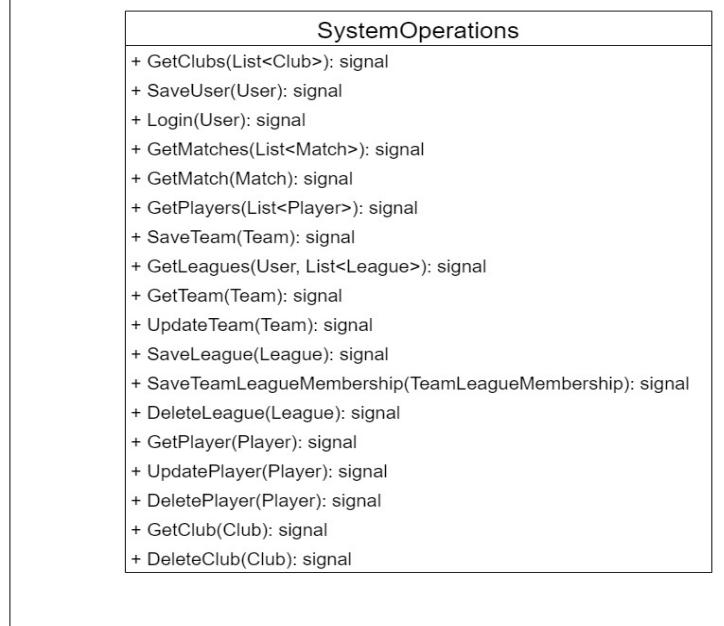
Табела Substitution		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	
	matchEventId	long	not null			INSERT RESTRICTED MatchEvent, Player
	matchId	long	not null			UPDATE RESTRICTED MatchEvent, Player
	gameweekId	long	not null			DELETE RESTRICTED MatchEvent
	outPlayerId	long	not null			

Табела 18. Табела Substitution

Како резултат фазе анализе добијени су логичка структура и понашање софтверског система приказани на слици 29.



Понашање система



Слика 29. Логичка структура и понашање софтверског система

4.3. Пројектовање

У овом одељку биће дат приказ архитектуре софтверског система тј. физичке структуре и понашања система. Користиће се класична тронивојска архитектура која се састоји из следећих нивоа: [4]

1. **Кориснички интерфејс** - Улазно-излазна репрезентација софтверског система.
2. **Апликациона логика** - Описује структуру и понашање софтверског система.
3. **Складиште података** - Чува стања атрибута софтверског система

Повезаност између нивоа софтверског система је приказана на слици 30. [4]



Слика 30. Тронивојска архитектура [4]

4.3.1. Пројектовање корисничког интерфејса

Кориснички интерфејс представља реализацију улаза и/или излаза софтверског система и састоји се од: [4]

1. Екранске форме
2. Контролера корисничког интерфејса

Приказ структуре корисничког интерфејса дат је на слици 31.



Слика 31. Структура корисничког интерфејса [4]

Кориснички интерфејс је дефинисан преко скупа екранских форми. Сценарија коришћења екранских форми су директно повезана са сценаријима случајева коришћења.

Постоје два аспекта пројектовања екранске форме: [4]

- a) Пројектовање сценарија СК који се изводе преко екранске форме.
- b) Пројектовање метода екранске форме.

Екранске форме клијентског дела програма су имплементиране помоћу радног оквира Angular, уз додатак Angular Material и Bootstrap библиотека. У даљем раду биће приказано пројектовање сценарија СК који се изводе преко екранских форми клијентског дела програма.

СК1: Случај коришћења – Регистрација корисника

Назив СК

Регистрација корисника

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за регистрацију. Учитана је листа клубова.

Football Fantasy

LINGLONG TIRE Super

Login Register

1 Personal Details 2 Favourite Club 3 Done

Personal Details

Username	Password
Username *	Password *
First Name	Last Name
First Name *	Last Name *
Email	Phone Number
Email *	Phone Number
Date of Birth	Gender
Choose a date *	<input checked="" type="radio"/> Male <input type="radio"/> Female

Next

Слика 32. Форма за регистрацију

Основни сценарио СК

- Корисник уноси корисничке податке. (АПУСО)

The screenshot shows a registration form divided into three main sections:

- Step 1: Personal Details** (highlighted in blue):
 - Username: user123
 - Password: (redacted)
 - First Name: Test
 - Last Name: User
 - Email: test@gmail.com
 - Phone Number: 061234567
 - Date of Birth: 10/23/1987
 - Gender: Male (radio button selected)
- Step 2: Favourite Club** (highlighted in blue):
 - A grid of 20 football club logos. The logo for "Voždovac" is highlighted with a green border.
- Step 3: Done** (highlighted in blue):
 - A confirmation message: "Please, confirm registration."
 - Buttons: Back, Register (highlighted in blue), and Done.

Слика 33. Исправно попуњена форма за регистрацију

- Корисник контролише да ли је коректно унео корисничке податке. (АНСО)

- Корисник позива систем да запамти корисничке податке. (АПСО)

Опис акције: корисник кликом на дугме *Register* позива системску операцију *SaveUser(User)*

- Систем памти податке о кориснику. (СО)

- Систем приказује кориснику поруку: "Registration completed!". (ИА)

Registration completed!

Слика 34. Порука о успешној регистрацији

Алтернативна сценарија

- Уколико систем не може да запамти податке о кориснику он приказује кориснику поруку: "Error!". (ИА)

Error!

Слика 35. Порука о неуспешној регистрацији

СК2: Случај коришћења - Пријављивање на систем

Назив СК

Пријављивање на систем

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за пријављивање.

Слика 36. Форма за пријављивање

Основни сценарио СК

1. Актор уноси корисничко име и лозинку. (АПУСО)

Слика 37. Исправно попуњена форма за пријављивање

2. Актор контролише да ли је коректно унео корисничко име и лозинку. (АНСО)

3. Актор позива систем да провери унете податке. (АПСО)

Опис акције: актор кликом на дугме Login позива системску операцију Login(User)

4. Систем проверава податке о актору. (СО)

5. Систем приказује актору поруку: "Login successful!". (ИА)



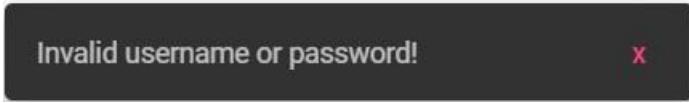
Login successful!

X

Слика 38. Порука о успешном пријављивању на систем

Алтернативна сценарија

- 5.1. Уколико **систем** не може да нађе **актора** он приказује **актору** поруку: “Invalid username or password!”. (ИА)



Invalid username or password!

X

Слика 39. Порука о неуспешном пријављивању на систем

СК3: Случај коришћења – Преглед меча

Назив СК

Преглед меча

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са мечевима. Учитана је листа мечева.

		Gameweek 7				
11-09-2020	Stadion na Banovom brdu	Čukarički	 1–3		Red Star Belgrade	▼
12-09-2020	Gradski stadion	Spartak Subotica	 1–1		Novi Pazar	▼
12-09-2020	Gradski stadion	TSC Bačka Top	 1–3		Radnički Niš	▼
12-09-2020	Stadion Surdulica	Radnik Surdulica	 0–2		Indija	▼
12-09-2020	Stadion Event Place	Voždovac	 4–0		Metalac GM	▼
12-09-2020	Stadion Partizana	Partizan	 3–0		Rad	▼

Слика 40. Форма за рад са мечевима

Основни сценарио СК

1. Корисник бира меч којем жели да приступи. (АПУСО)
2. Корисник позива систем да учита податке о одабраном мечу. (АПСО)
Опис акције: корисник кликом на ред жељеног меча позива системску операцију `GetMatch(Match)`
3. Систем учитава податке о одабраном мечу. (СО)
4. Систем обавештава корисника о успешном учитавању података о мечу поруком: “Selected match has been loaded!” и приказује податке о одабраном мечу. (ИА)

12-09-2020

Stadion Event Place

Voždovac



4-0



Metalac GM



Match Summary

Marko Putinčanin			9' 1:0	
Miloš Pantović			17' 2:0	
			35' 2:0	Ivan Rogač
Edin Ajdinović			41' 3:0	
			46' 3:0	Ljubiša Pecelj for Aleksandar Desančić
			46' 3:0	Nikola Popović for Ivan Rogač
Jovan Nišić for Miloš Stojčev			56' 3:0	
Jovan Nišić			58' 4:0	
Dragan Stoislavljević for Nikola Vujnović			62' 4:0	
			69' 4:0	Veljko Mijailovic for Prestige Mboungou
			69' 4:0	Ilija Milićević for Filip Antonijević
			69' 4:0	Marko Zočević for Jovan Kokir
			73' 4:0	Milos Vranjanin
Miloš Milović			74' 4:0	
Ivan Đorić for Ivan Milosavljević			75' 4:0	
Luka Cvetičanin for Miloš Pantović			75' 4:0	

Слика 41. Преглед меча

Алтернативна сценарија

4.1. Уколико систем не може да учита податке о мечу систем приказује кориснику поруку: "Error!". (ИА)



Слика 42. Порука о неуспешном прегледу меча

СК4: Случај коришћења – Креирање тима (Сложен СК)

Назив СК

Креирање тима

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за креирање тима. Учитана је листа играча.

Goalkeepers	Price	Points
Petrović	6.8	88
Borjan	6.4	83
Petrić	6.8	83
Popović	7.2	63
Drobnjak	5	59

Defenders	Price	Points
Pankov	6.5	97
Vujnović	6.8	95
Kamenović	7	89
Andrić	5.7	75
Živković	5.8	73

Midfielders	Price	Points
Makarić	11.3	122
Lukić	10.6	106
Tufegdžić	9.6	94
Birmančević	9.8	90
Kajević	9.4	85

Forwards	Price	Points
Asano	8.5	79
Katanic	7.7	64
Bojović	6.8	54
Mame	7.2	53
Putinčanin	8	53

Слика 43. Форма за креирање тима

Основни сценарио СК

1. Корисник уноси податке о тиму. (АПУСО)

Players Selected: 15/15

Money Remaining: 7.8

Goalkeepers	Price	Points
Petrović	6.8	88
Kamenović	6.2	83
Borjan	6.4	83
Petrić	6.8	83
Popović	7.2	63
Drobnjak	5	59
Vujnović	6.8	95
Kamenović	7	89
Andrić	5.7	75
Đurović	5.8	73

Defenders	Price	Points
Pankov	6.5	97
Vujnović	6.8	95
Kamenović	7	89
Andrić	5.7	75
Đurović	5.8	73

Midfielders	Price	Points
Makarić	11.3	122
Lukić	10.6	106
Tufegdzic	9.6	94
Birmancic	9.8	90
Kajevic	9.4	85

Forwards	Price	Points
Asano	8.5	79
Katanic	7.7	64
Bojovic	6.8	54
Mame	7.2	53
Putinčanin	8	53

Players Selected: 15/15

Money Remaining: 7.8

Save Team

Team Name *: FON Team

Captain *: Seydouba Soumah

Vice Captain *: Filip Pavišić

Cancel Save >

Goalkeepers	Price	Points
Petrović	6.8	88
Kamenović	6.2	83
Borjan	6.4	83
Petrić	6.8	83
Popović	7.2	63
Drobnjak	5	59
Vujnović	6.8	95
Kamenović	7	89
Andrić	5.7	75
Đurović	5.8	73

Defenders	Price	Points
Pankov	6.5	97
Vujnović	6.8	95
Kamenović	7	89
Andrić	5.7	75
Đurović	5.8	73

Midfielders	Price	Points
Makarić	11.3	122
Lukić	10.6	106
Tufegdzic	9.6	94
Birmancic	9.8	90
Kajevic	9.4	85

Forwards	Price	Points
Asano	8.5	79
Katanic	7.7	64
Bojovic	6.8	54
Mame	7.2	53
Putinčanin	8	53

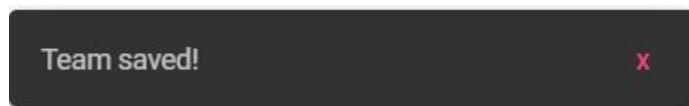
Слика 44. Исправно унети подаци о тиму

2. Корисник контролише да ли је коректно унео податке о тиму. (АНСО)

3. Корисник позива систем да запамти податке о тиму. (АПСО)

Опис акције: корисник кликом на дугме Save позива системску операцију SaveTeam(Team)

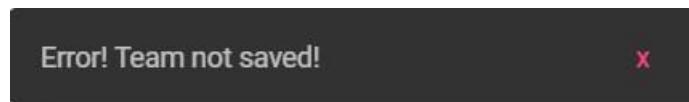
4. Систем **памти** податке о **тиму**. (СО)
5. Систем **приказује кориснику** поруку: "Team saved!". (ИА)



Слика 45. Порука о успешном креирању тима

Алтернативна сценарија

- 5.1. Уколико **систем** не може да запамти податке о **тиму** он приказује **кориснику** поруку: "Error! Team not saved!". (ИА)



Слика 46. Порука о неуспешном креирању тима

СК5: Случај коришћења – Преглед тима

Назив СК

Преглед тима

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са лигама. Учитана је листа лига.

The screenshot shows the Football Fantasy application's interface. At the top, there is a navigation bar with tabs: Team, Points, Transfers, and Leagues (which is highlighted). On the right side of the header, there are buttons for ADMIN (with a user icon), Logout, and other user-related options. The main content area has a purple header with the text "Football Fantasy". Below it, there are two sections: "Leagues" and "Create League". The "Leagues" section contains a table with the following data:

ID	Name	Owner	Action
2	liga liga	Admin Admin	Details Delete
3	nova liga	Admin Admin	Hide Delete
4	zmajevl	Admin Admin	Details Delete
1	liga	Admin Admin	Details Leave
6	naziv lige	Admin Admin	Details Delete

Below this table, there is a small pagination indicator: "1 - 5 of 7" with arrows for navigating through the pages. To the right of the leagues table is a "Create League" form with a text input field for "League Name *". Below it is a "Create" button. Underneath the "Create League" section is another section titled "Join League" with a text input field for "League ID *". Below it is a "Join" button. At the bottom of the main content area, there is a sub-section titled "nova liga" containing a table with the following data:

Rank	Team	User	Points
1	timcina	Admin Admin	683
2	tim	admin admin	451

Слика 47. Форма за рад са лигама

Основни сценарио СК

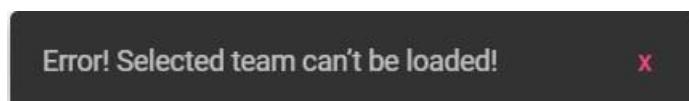
1. Актор бира тим којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном тиму. (АПСО)
Опис акције: актор кликом на ред жељеног тима позива системску операцију `GetTeam(Team)`
3. Систем учитава податке о одабраном тиму. (СО)
4. Систем обавештава актора о успешном учитавању података о тиму поруком: "Selected team has been loaded!" и приказује податке о одабраном тиму. (ИА)



Слика 48. Преглед тима

Алтернативна сценарија

4.1. Уколико **систем** не може да учита податке о **тиму систем** приказује **актору** поруку: "Error! Selected team can't be loaded!". (ИА)



Слика 49. Порука о неуспешном прегледу тима

СК6: Случај коришћења – Измена тима

Назив СК

Измена тима

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са тимом. Учитани су тим и листа играча.

The screenshot shows the Football Fantasy application interface. At the top, there is a navigation bar with tabs: Team (selected), Points, Transfers, and Leagues. On the far right of the bar is a Logout link. Below the navigation bar, the title "Football Fantasy" is displayed in a large, bold font. To the right of the title are two logos: "LINGLONG TIRE" and "Supreme".

The main area features a green football pitch diagram representing a 4-3-3 team formation. The players are listed with their names: Petrović (Goalkeeper), Kamenović, Andrić, Ostojačić, Pavilić (defenders); Netkō, Janjić, Radivojević, Soumah (midfielders); and Asano, Lasicas (attackers). Below the pitch diagram is a row of four player cards: Drobnič, Gajić, Đurić, and Oumarou.

To the right of the pitch diagram is a "Substitution" section. It contains two icons of players, one labeled "Select player" and the other "Save". Between them is a red arrow pointing left and a green arrow pointing right, indicating the direction of substitution. Below these arrows are "Cancel" and "Switch" buttons.

On the right side of the screen, there is a "Team Details" section. It includes a "Captain" card (Seydouba Soumah) and a "Vice Captain" card (Filip Pavilić). Below these cards, the team's statistics are listed:

Team:	FON Team
User:	Test User
Total Points:	0
Free Transfers:	2
Team Value:	92.2
Money Remaining:	7.8

Below the statistics are two more sections: "GW Stats" and "Leagues". The "GW Stats" section has columns for GW, Points, and a status indicator (0 of 1). The "Leagues" section has columns for ID, Name, and a status indicator (0 of 1).

Слика 50. Форма за рад са тимом

Основни сценарио СК

- Корисник мења податке о тиму. (АПУСО)

Substitution

Marko Gajić → Miloš Ostojić

Team Details

Captain Seydouba Soumah	Vice Captain Filip Pavišić
Team: User: Total Points: Free Transfers: Team Value: Money Remaining:	FON Team Test User 0 2 92.2 7.8
Favourite Club 	

GW Stats

GW	Points
0 of n	< >

Leagues

ID	Name
0 of n	< >

Слика 51. Измена тима

- Корисник контролише да ли је коректно унео податке о тиму. (АНСО)
- Корисник позива систем да запамти податке о тиму. (АПСО)

Опис ације: корисник кликом на дугме Save позива системску операцију `UpdateTeam(Team)`
- Систем памти податке о тиму. (СО)
- Систем приказује кориснику поруку: "Team saved!". (ИА)

Алтернативна сценарија

- Уколико систем не може да запамти податке о тиму он приказује кориснику поруку: "Team not saved!". (ИА)



Слика 52. Порука о неуспешној измени тима

СК7: Случај коришћења – Креирање лиге

Назив СК

Креирање лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама.

Основни сценарио СК

1. Корисник уноси податке о лиги. (АПУСО)

Create League

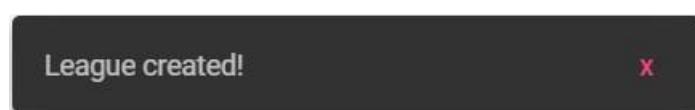
League Name *

FON League

Create

Слика 53. Исправно унети подаци о лиги

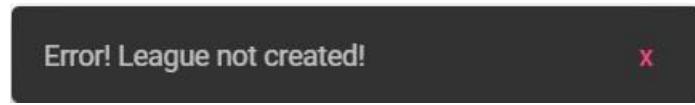
2. Корисник контролише да ли је коректно унео податке о лиги. (АНСО)
3. Корисник позива систем да запамти податке о лиги. (АПСО)
Опис акције: корисник кликом на дугме Create позива системску операцију SaveLeague(League)
4. Систем памти податке о лиги. (СО)
5. Систем приказује кориснику поруку: "League created!". (ИА)



Слика 54. Порука о успешном креирању лиге

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о лиги он приказује кориснику поруку: "Error! League not created!". (ИА)



Слика 55. Порука о неуспешном креирању лиге

СК8: Случај коришћења – Креирање чланства лиге

Назив СК

Креирање чланства лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама.

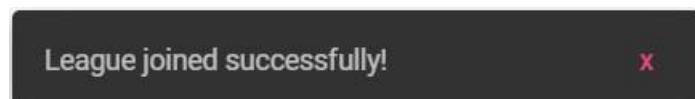
Основни сценарио СК

1. Корисник уноси податке о чланству. (АПУСО)

The screenshot shows a user interface for joining a league. At the top, it says 'Join League'. Below that is a form field labeled 'League ID *' with the value '4' entered. At the bottom is a large blue button labeled 'Join'.

Слика 56. Исправно унети подаци за приступање лиги

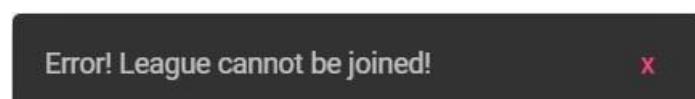
2. Корисник контролише да ли је коректно унео податке о чланству. (АНСО)
3. Корисник позива систем да запамти податке о чланству. (АПСО)
Опис акције: корисник кликом на дугме Join позива системску операцију SaveTeamLeagueMembership(TeamLeagueMembership)
4. Систем памти податке о чланству. (СО)
5. Систем приказује кориснику поруку: "League joined successfully!". (ИА)



Слика 57. Порука о успешном приступању лиги

Алтернативна сценарија

- 5.1. Уколико систем не може да запамти податке о чланству он приказује кориснику поруку: "Error! League cannot be joined!". (ИА)



Слика 58. Порука о неуспешном приступању лиги

СК9: Случај коришћења – Брисање лиге

Назив СК

Брисање лиге

Актори СК

Корисник

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује форму за рад са лигама. Учитана је листа лига.

Основни сценарио СК

1. Корисник бира лигу коју жели да обрише. (АПУСО)

Leagues

ID	Name	Owner	Action	
11	FON League	✓	Details	Delete
12	Test League 1	✓	Details	Delete
13	Test League 2	✓	Details	Delete
14	Test League 3	✓	Details	Delete

1 – 4 of 4

< >

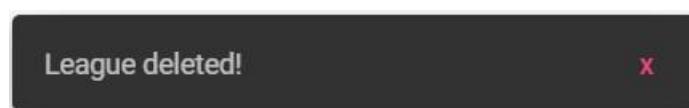
Слика 59. Брисање лиге

2. Корисник позива систем да обрише изабрану лигу. (АПСО)

Опис акције: корисник кликом на дугме Delete позива системску операцију DeleteLeague(League)

3. Систем брише лигу. (СО)

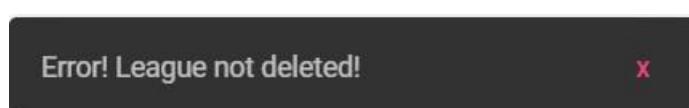
4. Систем приказује кориснику поруку: "League deleted!". (ИА)



Слика 60. Порука о успешном брисању лиге

Алтернативна сценарија

- 4.1. Уколико систем не може да обрише лигу он приказује кориснику поруку: "Error! League not deleted!". (ИА)



Слика 61. Порука о неуспешном брисању лиге

СК10: Случај коришћења – Преглед играча

Назив СК

Преглед играча

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитана је листа играча.

Image	Name	Nationality	Birth Date	Age	Position	Height	Weight	Club	Select club
	Saša Zdjelar	SRB	1995-03-20	25	MF	183cm	75kg	Partizan	<input type="radio"/> All
	Takuma Asano	JPN	1994-11-10	26	FW	171cm	70kg	Partizan	<input type="radio"/> Red Star Belgrade
	Filip Stevanovic	SRB	2002-09-25	18	FW			Partizan	<input checked="" type="radio"/> Partizan
	Aleksandar Popović	SRB	1983-11-02	37	GK	180cm		Partizan	<input type="radio"/> Vojvodina
	Slobodan Urošević	SRB	1994-04-15	26	DF	183cm		Partizan	<input type="radio"/> Proleter Novi Sad
	Macky Bagnack	CMR	1995-06-07	25	DF	182cm		Partizan	<input type="radio"/> Spartak Subotica
	Bibras Natkho	ISR	1988-02-18	32	MF	175cm	73kg	Partizan	<input type="radio"/> Čukarički
	Aleksandar Miljković	SRB	1990-02-26	30	DF	181cm	76kg	Partizan	<input type="radio"/> Voždovac
	Seydouba Soumah	GUI	1991-06-11	29	MF	161cm		Partizan	<input type="radio"/> Mladost Lučani
									<input type="radio"/> Radnički Niš
									<input type="radio"/> Metalac GM
									<input type="radio"/> TSC Bačka Top
									<input type="radio"/> Radnik Surđulica
									<input type="radio"/> Indija
									<input type="radio"/> Javor Ivanjica
									<input type="radio"/> Novi Pazar
									<input type="radio"/> FK Zlatibor Čajetina
									<input type="radio"/> Napredak Kruševac
									<input type="radio"/> Rad
									<input type="radio"/> Mačva Šabac
									<input type="radio"/> Bačka Bačka Palanka
									<input type="radio"/> Zlatibor Čajetina
									<input type="radio"/> OFK Bačka

Слика 62. Форма за рад са играчима

Основни сценарио СК

1. Актор бира играча којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном играчу. (АПСО)
Опис акције: актор кликом на ред жељеног играча позива системску операцију `GetPlayer(Player)`
3. Систем учитава податке о одабраном играчу. (СО)

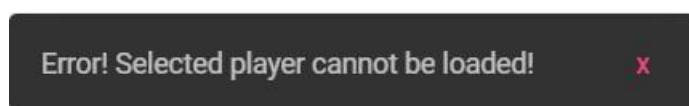
4. Систем обавештава актора о успешном учитавању података о играчу поруком: "Selected player has been loaded!" и приказује податке о одабраном играчу. (ИА)

The screenshot shows a user interface for managing players. On the left, there is a grid of player cards with columns for Image, Name, Nationality, Birth Date, Age, Position, Height, Weight, and Club. A filter bar at the top left allows searching by name. On the right, a modal window titled "Player Details" displays the selected player's information. The player's name is Saša Zdjelar, nationality is SRB, birth date is 3/20/1995, age is 25, position is MF, height is 183cm, weight is 75kg, and club is Partizan. The background shows a sidebar titled "Select club" with various club names listed as radio buttons, and the "Partizan" button is selected.

Слика 63. Преглед играча

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о играчу систем приказује актору поруку: "Error! Selected player cannot be loaded!". (ИА)



Слика 64. Порука о неуспешном учитавању играча

СК11: Случај коришћења – Измена играча

Назив СК

Измена играча

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитане су листе играча и клубова.

Основни сценарио СК

1. Администратор бира играча којем жели да приступи. (АПУСО)
2. Администратор позива систем да учита податке о одабраном играчу. (АПСО)
Опис акције: администратор кликом на ред жељеног играча позива системску операцију *GetPlayer(Player)*
3. Систем учитава податке о одабраном играчу. (СО)
4. Систем обавештава администратора о успешном учитавању података о играчу поруком: "Selected player has been loaded!" и приказује податке о одабраном играчу. (ИА)

Player Details

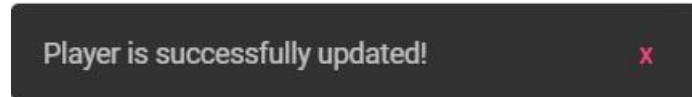
The screenshot shows a player details form for a player named 'Saša Zdjelar'. The form includes fields for Name, Image, Nationality, Birth Date, Age, Position, Height, Weight, and Club. The player's name is displayed prominently at the top right. The form is a standard web-based input field.

Name *	Saša Zdjelar
Image	https://fbref.com/req/202005121/images/headshots/bbddc31f_2018.jpg
Nationality	SRB
Birth Date	3/20/1995
Age	25
Position *	MF
Height	183cm
Weight	75kg
Club *	Partizan

Cancel Update Delete

Слика 65. Измена играча

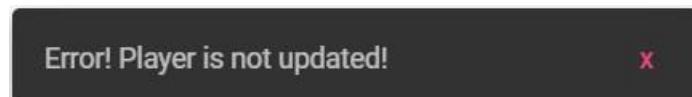
5. Администратор мења податке о играчу. (АПУСО)
6. Администратор контролише да ли је коректно унео податке о играчу. (АНСО)
7. Администратор позива систем да запамти податке о играчу. (АПСО)
Опис акције: администратор кликом на дугме Update позива системску операцију UpdatePlayer(Player)
8. Систем памти податке о играчу. (СО)
9. Систем приказује администратору поруку: "Player is successfully updated!". (ИА)



Слика 66. Порука о успешној измени играча

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о играчу систем приказује администратору поруку: "Error! Selected player cannot be loaded!". Прекида се извршење сценарија. (ИА)
- 9.1. Уколико систем не може да запамти податке о играчу он приказује администратору поруку: "Error! Player is not updated!". (ИА)



Слика 67. Порука о неуспешној измени играча

СК12: Случај коришћења – Брисање играча

Назив СК

Брисање играча

Актори СК

Администратор

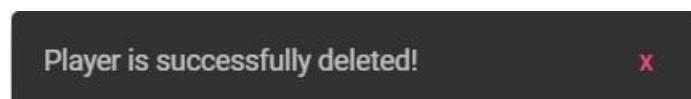
Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са играчима. Учитане су листе играча и клубова.

Основни сценарио СК

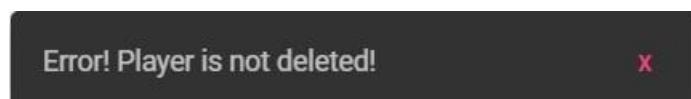
1. Администратор бира играча којег жели да обрише. (АПУСО)
2. Администратор позива систем да учита податке о одабраном играчу. (АПСО)
Опис акције: администратор кликом на ред жељеног играча позива системску операцију *DeletePlayer(Player)*
3. Систем учитава податке о одабраном играчу. (СО)
4. Систем обавештава администратора о успешном учитавању података о играчу поруком: “Selected player has been loaded!” и приказује податке о одабраном играчу. (ИА)
5. Администратор позива систем да обрише изабраног играча. (АПСО)
Опис акције: администратор кликом на дугме *Delete* позива системску операцију *DeletePlayer(Player)*
6. Систем брише играча. (СО)
7. Систем приказује администратору поруку: “Player is successfully deleted!”. (ИА)



Слика 68. Порука о успешном брисању играча

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о играчу систем приказује администратору поруку: “Error! Selected player cannot be loaded!”. Прекида се извршење сценарија. (ИА)
- 7.1. Уколико систем не може да запамти податке о играчу он приказује администратору поруку: “Error! Player is not deleted!”. (ИА)



Слика 69. Порука о неуспешном брисању играча

СК13: Случај коришћења – Преглед клуба

Назив СК

Преглед клуба

Актори СК

Корисник, администратор

Учесници СК

Актор и систем (програм)

Предуслов: Систем је укључен и актор је пријављен под својом шифром. Систем приказује форму за рад са клубовима. Учитана је листа клубова.

<p>Red Star Belgrade Manager: Dejan Stanković</p>  <p>UPDATE DELETE</p>	<p>Partizan Manager: Aleksandar Stanojević</p>  <p>UPDATE DELETE</p>	<p>Vojvodina Manager: Nenad Lalatović</p>  <p>UPDATE DELETE</p>	<p>Proleter Novi Sad Manager: Branko Žigić</p>  <p>UPDATE DELETE</p>
<p>Spartak Subotica Manager: Serbia rs</p>  <p>UPDATE DELETE</p>	<p>Čukarički Manager: Serbia rs</p>  <p>UPDATE DELETE</p>	<p>Voždovac Manager: Jovan Damjanović</p>  <p>UPDATE DELETE</p>	<p>Mladost Lučani Manager: Nenad Milovanović</p>  <p>UPDATE DELETE</p>
<p>Radnički Niš Manager: Vladimir Gačinović</p>  <p>UPDATE DELETE</p>	<p>Metalac GM Manager: Žarko Lazetić</p>  <p>UPDATE DELETE</p>	<p>TSC Bačka Top Manager: Mladen Krstajić</p>  <p>UPDATE DELETE</p>	<p>Radnik Surđulica Manager: Slavoljub Đorđević</p>  <p>UPDATE DELETE</p>
<p>Indija Manager: Serbia rs</p>  <p>UPDATE DELETE</p>	<p>Javor Ivanjica Manager: Igor Bondžulić</p>  <p>UPDATE DELETE</p>	<p>Novi Pazar Manager: Radoslav Batak</p>  <p>UPDATE DELETE</p>	<p>FK Zlatibor Čajetina Manager: Zoran Njeguš</p>  <p>UPDATE DELETE</p>

Слика 70. Форма за рад са клубовима

Основни сценарио СК

1. Актор бира клуб којем жели да приступи. (АПУСО)
2. Актор позива систем да учита податке о одабраном клубу. (АПСО)
Опис акције: актор кликом на лого жељеног клуба позива системску операцију `GetClub(Club)`
3. Систем учитава податке о одабраном клубу. (СО)

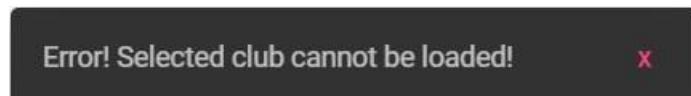
4. Систем обавештава актора о успешном учитавању података о клубу поруком: "Selected club has been loaded!" и приказује податке о одабраном клубу. (ИА)

Image	Name	Nationality	Birth Date	Age	Position	Height	Weight	Club
	Saša Zdjelar	SRB	1995-03-20	25	MF	183cm	75kg	Partizan
	Takuma Asano	JPN	1994-11-10	26	FW	171cm	70kg	Partizan
	Filip Stevanović	SRB	2002-09-25	18	FW			Partizan
	Aleksandar Popović	SRB	1983-11-02	37	GK	180cm		Partizan
	Slobodan Urošević	SRB	1994-04-15	26	DF	183cm		Partizan
	Macky Bagnack	CMR	1995-06-07	25	DF	182cm		Partizan
	Bilbas Natkho	ISR	1988-02-18	32	MF	175cm	73kg	Partizan
	Aleksandar Miljković	SRB	1990-02-26	30	DF	181cm	76kg	Partizan
	Seydouba Soumah	GUI	1991-06-11	29	MF	161cm		Partizan

Слика 71. Преглед клуба

Алтернативна сценарија

- 4.1. Уколико систем не може да учита податке о клубу систем приказује актору поруку: "Error! Selected club cannot be loaded!". (ИА)



Слика 72. Порука о неуспешном прегледу клуба

СК14: Случај коришћења – Брисање клуба

Назив СК

Брисање клуба

Актори СК

Администратор

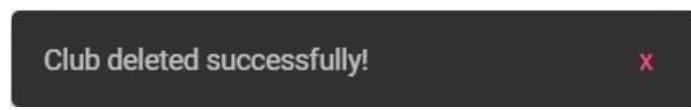
Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује форму за рад са клубовима. Учитана је листа клубова.

Основни сценарио СК

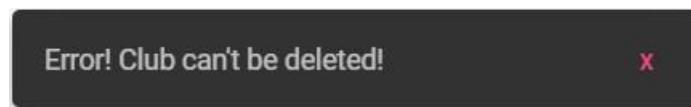
1. Администратор бира клуб који жели да обрише. (АПУСО)
2. Администратор позива систем да обрише изабрани клуб. (АПСО)
Опис акције: администратор кликом на дугме Delete позива системску операцију DeleteClub(Club)
3. Систем брише клуб. (СО)
4. Систем приказује администратору поруку: "Club deleted successfully!". (ИА)



Слика 73. Порука о успешном брисању клуба

Алтернативна сценарија

- 4.1. Уколико систем не може да обрише клуб он приказује администратору поруку: "Error! Club can't be deleted!". (ИА)



Слика 74. Порука о неуспешном брисању клуба

4.3.2. Пројектовање апликационе логике

У оквиру апликационе логике се пројектују контролер апликационе логике и пословна логика. [4]

4.3.2.1. Контролер апликационе логике

Контролер апликационе логике има улогу да прихвата све захтеве послате од стране клијентског дела апликације. Након пријема, контролер даље прослеђује захтев за извршење системске операције сервису који је одговоран за њено извршење а он даље до Repository класе која је одговорна за комуникацију са базом. Након извршења системске операције, резултат се преко контролера шаље назад до клијентске стране.

Потребно је да се контролер означи Spring анотацијом @RestController. Методе у оквиру контролера, у зависности од типа захтева, треба означити неком од следећих анотација:

1. @GetMapping
2. @PostMapping
3. @PutMapping
4. @DeleteMapping

Кључни параметар који треба подесити у оквиру наведених анотација јесте *value* параметар. Он садржи URI адресе на коју треба упутити захтев како би га означена метода контролера обрадила.

У наставку је дат пример програмског кода контролера:

```
@RestController
@RequestMapping("/teams")
public class TeamController {

    @Autowired
    TeamService teamService;

    @PostMapping(value = "/team")
    ResponseEntity<?> save(@RequestBody Team team) {
        return new ResponseEntity<>(teamService.save(team), HttpStatus.OK);
    }

    @GetMapping(value = "/all")
    ResponseEntity<?> findAll() {
        return new ResponseEntity<>(teamService.findAll(), HttpStatus.OK);
    }

    @GetMapping(value = "/team/{id}")
    ResponseEntity<?> findById(@PathVariable("id") Long id) {
        return new ResponseEntity<>(teamService.findById(id), HttpStatus.OK);
    }

    @GetMapping(value = "/user/{id}")
    ResponseEntity<?> findUserId(@PathVariable("id") Long userId) {
        return new ResponseEntity<>(teamService.findUserId(userId), HttpStatus.OK);
    }

    @GetMapping(value = "/league/{id}")
    ResponseEntity<?> findByLeagueId(@PathVariable("id") Long leagueId) {
        return new ResponseEntity<>(teamService.findByLeagueId(leagueId), HttpStatus.OK);
    }
}
```

4.3.2.2. Пословна логика

Пословна логика обухвата структуру и понашање система.

4.3.2.2.1. Пројектовање понашања софтверског система – системске операције

За сваки од претходно дефинисаних уговора пројектује се концептуално решење.

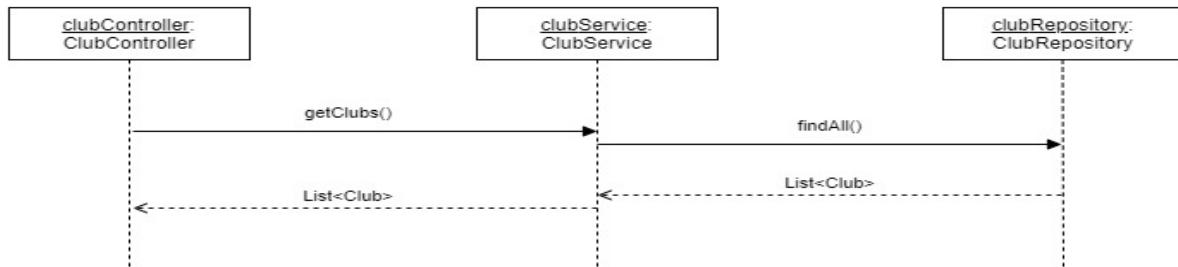
Уговор УГ1: GetClubs

Операција: **GetClubs(List<Club>): signal;**

Веза са СК: CK1, CK11, CK12, CK13, CK14

Предуслови: /

Постуслови: /



Дијаграм 32. Уговор УГ1: GetClubs

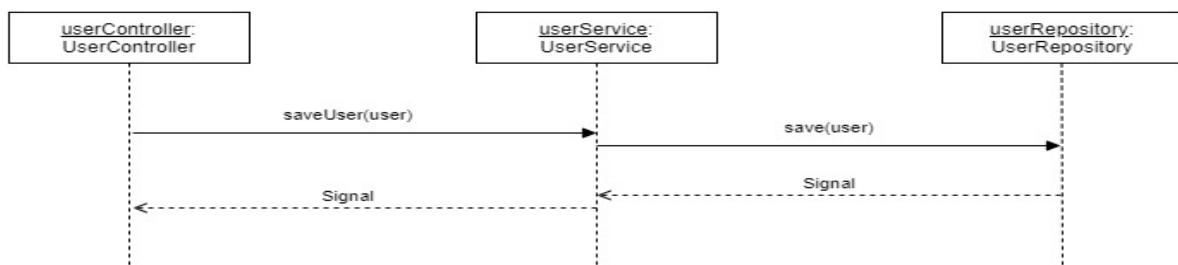
Уговор УГ2: SaveUser

Операција: **SaveUser(User): signal;**

Веза са СК: CK1

Предуслови: Вредносна и структурна ограничења над објектом User морају бити задовољена.

Постуслови: Креиран је нови корисник.



Дијаграм 33. Уговор УГ2: SaveUser

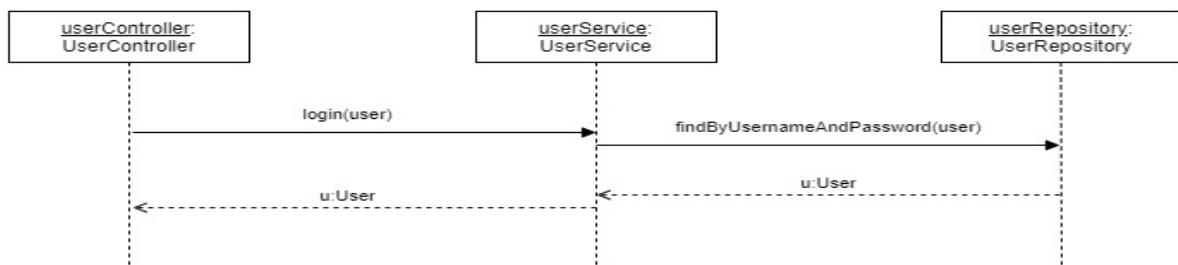
Уговор УГ3: Login

Операција: **Login(User): signal;**

Веза са СК: CK2

Предуслови: /

Постуслови: /



Дијаграм 34. Уговор УГ3: Login

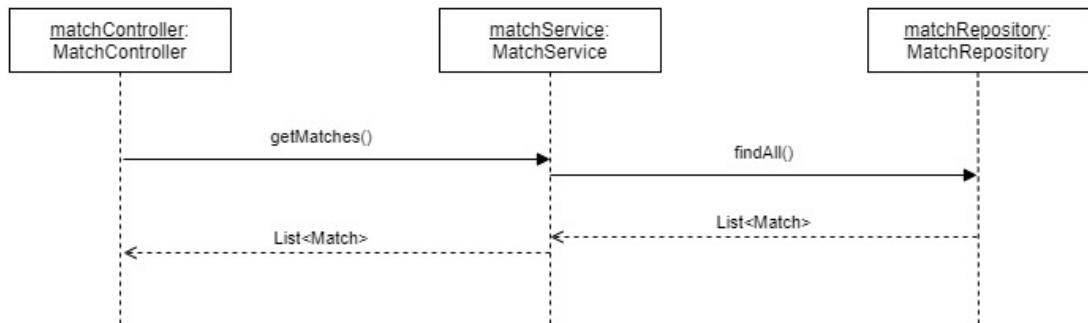
Уговор УГ4: GetMatches

Операција: **GetMatches(List<Match>): signal;**

Веза са СК: СК3

Предуслови: /

Постуслови: /



Дијаграм 35. Уговор УГ4: GetMatches

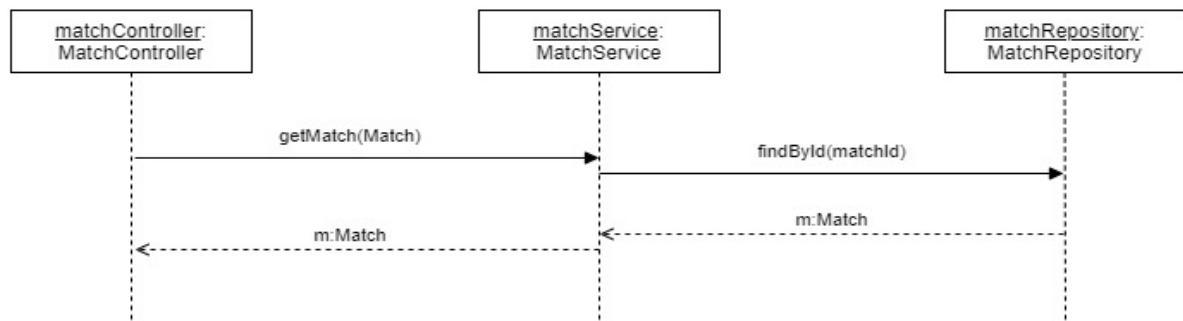
Уговор УГ5: GetMatch

Операција: **GetMatch(Match): signal;**

Веза са СК: СК3

Предуслови: /

Постуслови: /



Дијаграм 36. Уговор УГ5: GetMatch

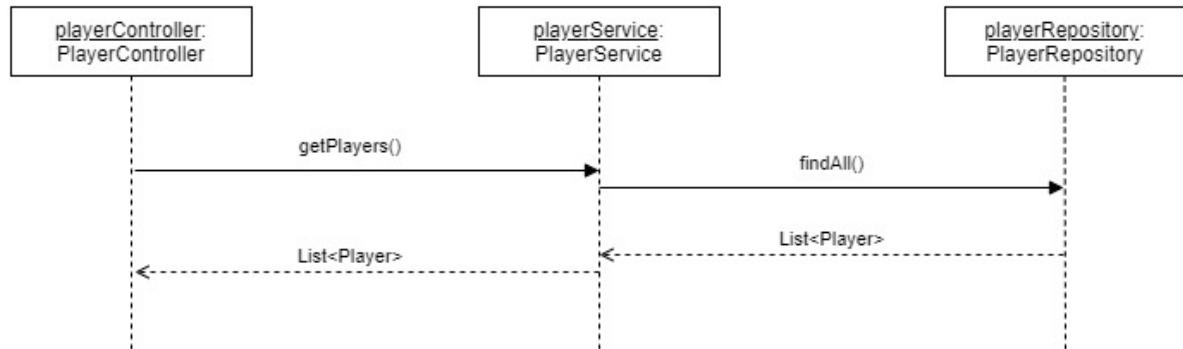
Уговор УГ6: GetPlayers

Операција: **GetPlayers(List<Player>): signal;**

Веза са СК: СК4, СК6, СК10, СК11, СК12

Предуслови: /

Постуслови: /



Дијаграм 37. Уговор УГ6: GetPlayers

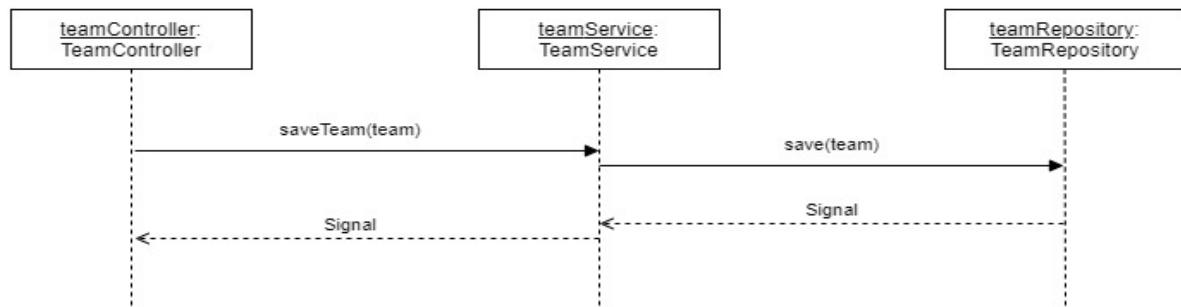
Уговор УГ7: SaveTeam

Операција: **SaveTeam(Team): signal;**

Веза са СК: СК4

Предуслови: Вредносна и структурна ограничења над објектом Team морају бити задовољена.

Постуслови: Креiran је нови тим.



Дијаграм 38. Уговор УГ7: SaveTeam

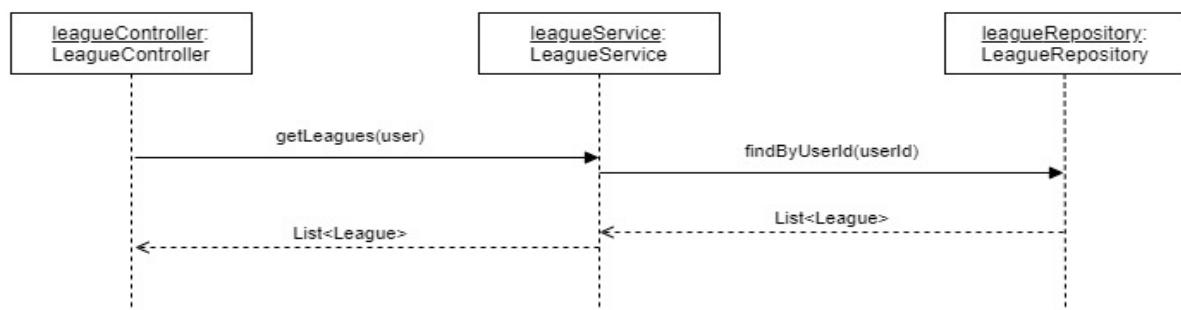
Уговор УГ8: GetLeagues

Операција: **GetLeagues(User, List<League>): signal;**

Веза са СК: СК5, СК9

Предуслови: /

Постуслови: /



Дијаграм 39. Уговор УГ8: GetLeagues

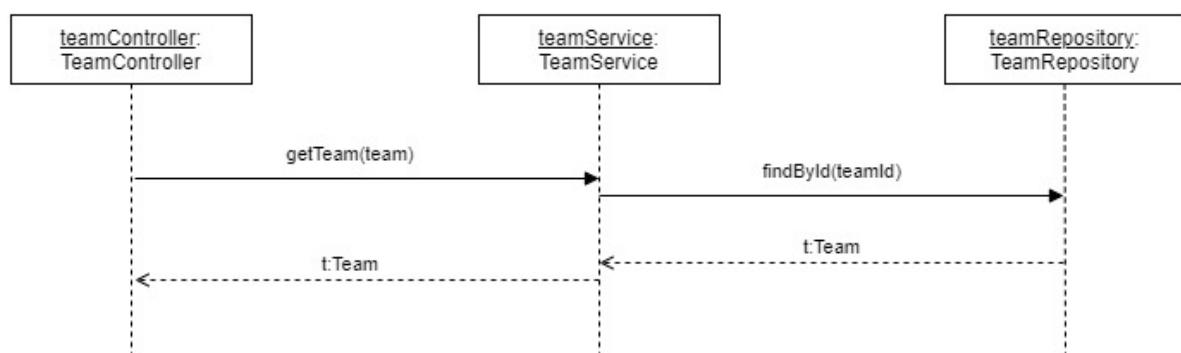
Уговор УГ9: GetTeam

Операција: **GetTeam(Team): signal;**

Веза са СК: СК5, СК6

Предуслови: /

Постуслови: /



Дијаграм 40. Уговор УГ9: GetTeam

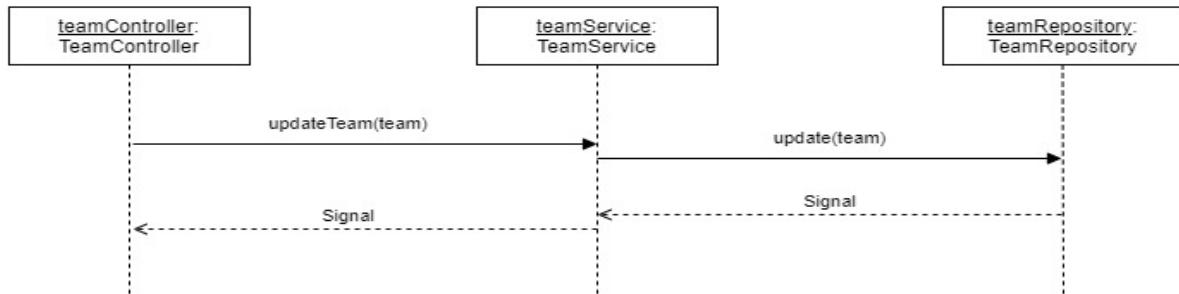
Уговор УГ10: UpdateTeam

Операција: **UpdateTeam(Team): signal;**

Веза са СК: СК6

Предуслови: Вредносна и структурна ограничења над објектом Team морају бити задовољена.

Постуслови: Тим је изменен.



Дијаграм 41. Уговор УГ10: *UpdateTeam*

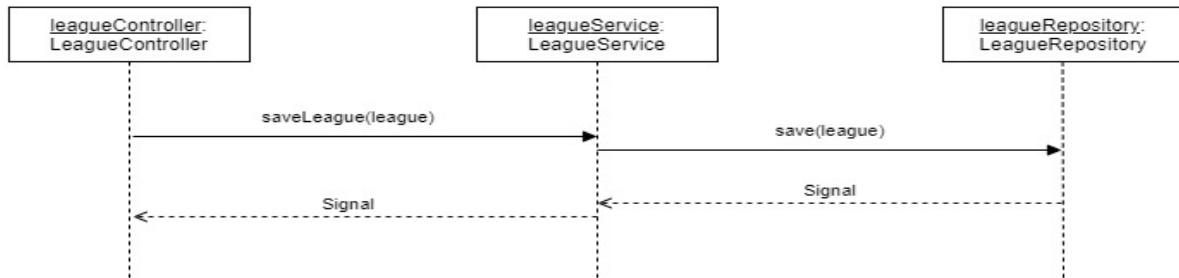
Уговор УГ11: SaveLeague

Операција: **SaveLeague(League): signal;**

Веза са СК: СК7

Предуслови: Вредносна и структурна ограничења над објектом League морају бити задовољена.

Постуслови: Креирана је нова лига.



Дијаграм 42. Уговор УГ11: *SaveLeague*

Уговор УГ12: SaveTeamLeagueMembership

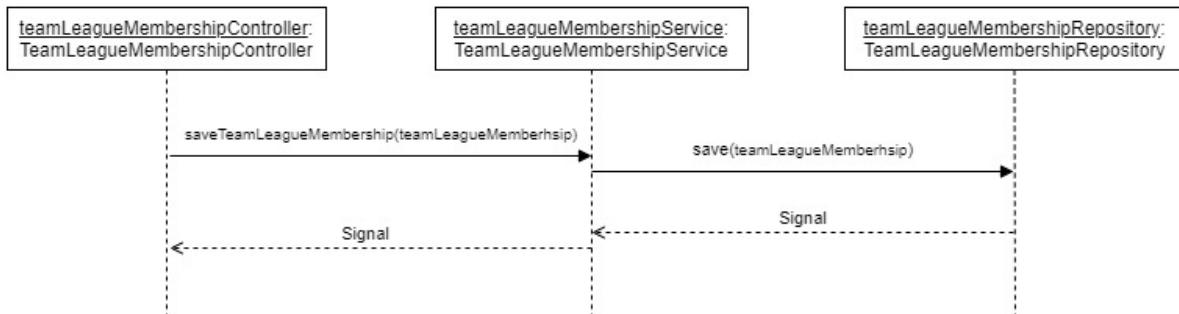
Операција: **SaveTeamLeagueMembership(TeamLeagueMembership): signal;**

Веза са СК: СК8

Предуслови: Вредносна и структурна ограничења над објектом

TeamLeagueMembership морају бити задовољена.

Постуслови: Креирано је ново чланство лиге.



Дијаграм 43. Уговор УГ12: *SaveTeamLeagueMembership*

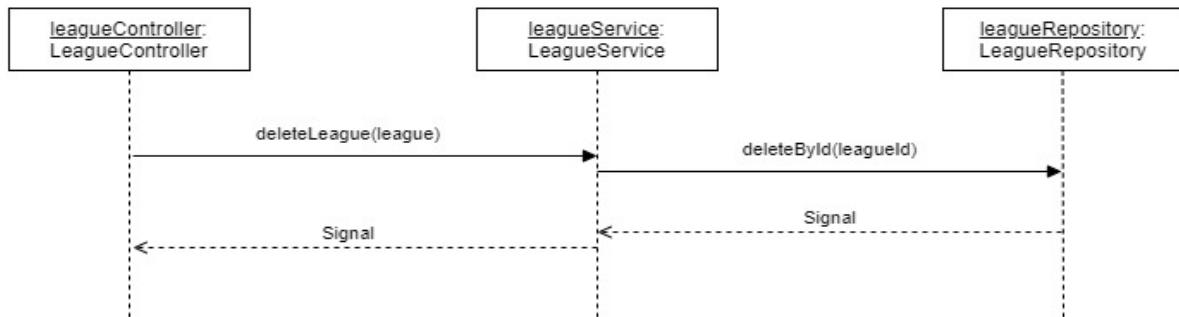
Уговор УГ13: DeleteLeague

Операција: **DeleteLeague(League): signal;**

Веза са СК: СК9

Предуслови: Структурна ограничења над објектом League морају бити задовољена.

Постуслови: Лига је обрисана.



Дијаграм 44. Уговор УГ13: DeleteLeague

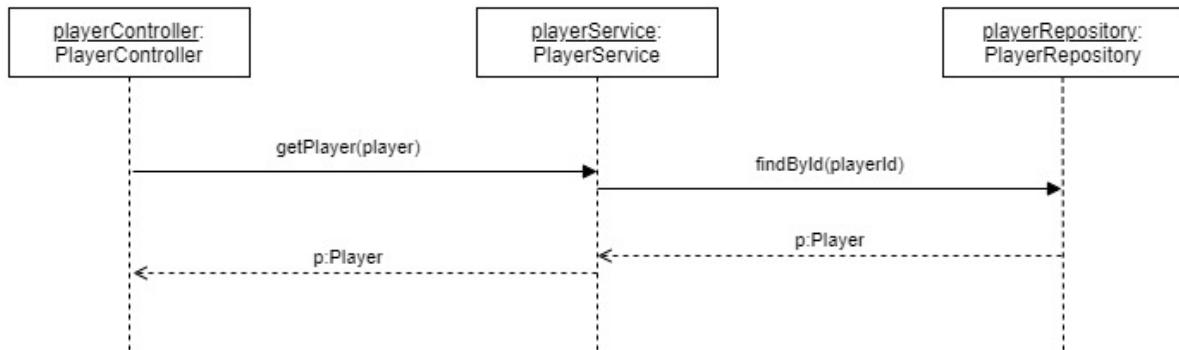
Уговор УГ14: GetPlayer

Операција: **GetPlayer(Player): signal;**

Веза са СК: СК10, СК11, СК12

Предуслови: /

Постуслови: /



Дијаграм 45. Уговор УГ14: GetPlayer

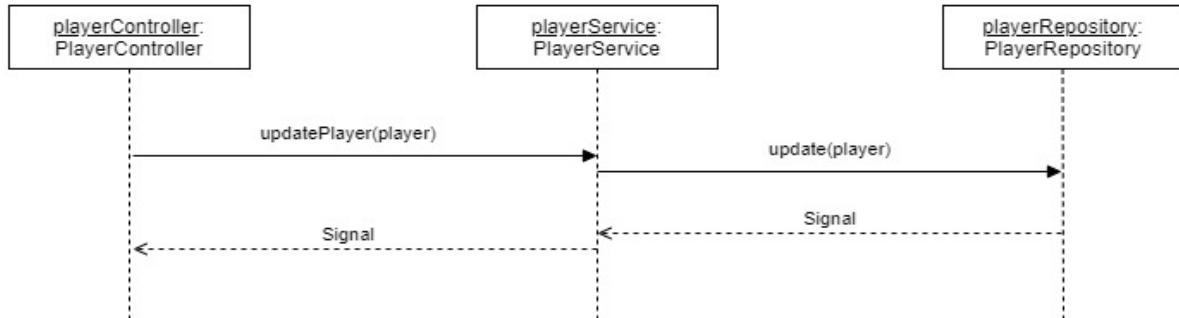
Уговор УГ15: UpdatePlayer

Операција: **UpdatePlayer(Player): signal;**

Веза са СК: СК11

Предуслови: Вредносна и структурна ограничења над објектом Player морају бити задовољена.

Постуслови: Играч је изменен.



Дијаграм 46. Уговор УГ15: UpdatePlayer

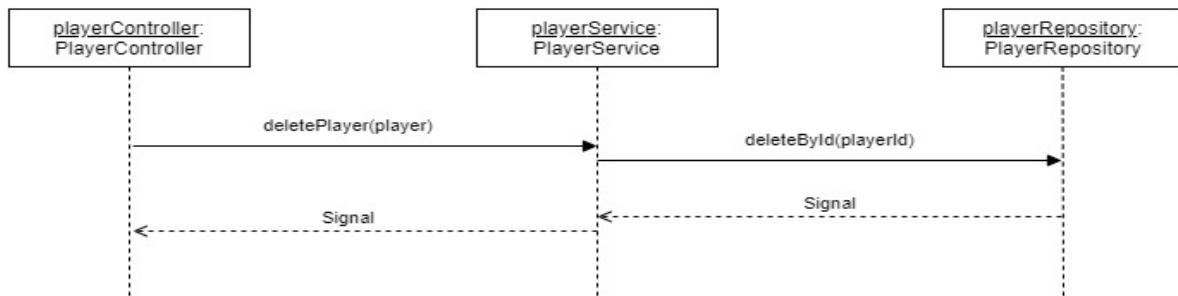
Уговор УГ16: DeletePlayer

Операција: DeletePlayer(Player); signal;

Веза са СК: CK12

Предуслови: Структурна ограничења над објектом Player морају бити задовољена.

Постуслови: Играч је обрисан.



Дијаграм 47. Уговор УГ16: DeletePlayer

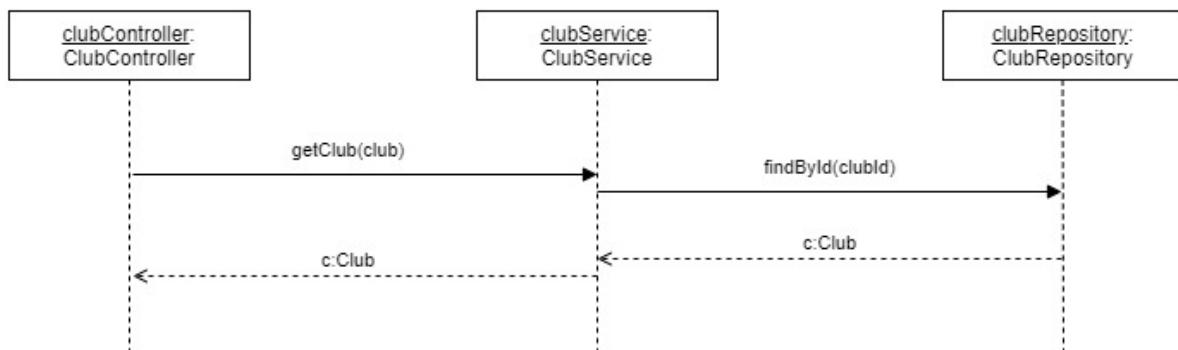
Уговор УГ17: GetClub

Операција: GetClub(Club); signal;

Веза са СК: CK13

Предуслови: /

Постуслови: /



Дијаграм 48. Уговор УГ17: GetClub

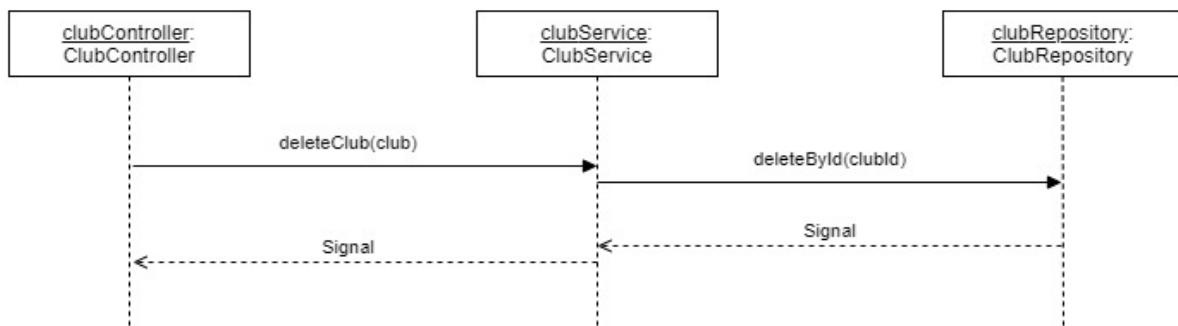
Уговор УГ18: DeleteClub

Операција: DeleteClub(Club); signal;

Веза са СК: CK14

Предуслови: Структурна ограничења над објектом Club морају бити задовољена.

Постуслови: Клуб је обрисан.



Дијаграм 49. Уговор УГ18: DeleteClub

4.3.2.2.2. Пројектовање структуре софтверског система

На основу концептуалних класа праве се софтверске класе структуре. Свака класа садржи getter, setter, `toString` методе, као и параметризоване и непараметризоване конструкторе.

4.3.3. Пројектовање брокера базе података

Брокер базе података је софтверска класа одговорна за комуникацију између пословне логике и складишта података. Другим речима, пројектује се како би се обезбедио перзистентни сервис објектима доменских класа који се чувају у бази података.

Spring Data, модул радног оквира Spring, омогућава једноставно креирање брокера базе података наслеђивањем интерфејса *CrudRepository*. Он укључује велики број основних операција над ентитетима, омогућава претрагу по резервисаним кључним речима или самостално писање упита.

У наставку је дат пример програмског кода Repository класе која има улогу брокера базе података:

```
public interface PlayerRepository extends CrudRepository<Player, Long> {

    Player findByUrl(String url);

    List<Player> findByOrderByClubNameAsc();

    @Modifying
    @Query(value = "UPDATE players SET total_points = :totalPoints, price =
    :price WHERE id = :id", nativeQuery = true)
    void updateTotalPointsAndPrice(@Param("id") Long id, @Param("totalPoints")
    int totalPoints, @Param("price") double price);

    List<Player> findByOrderByTotalPointsDesc();

    List<Player> findByClubId(Long clubId);

}
```

4.3.4. Пројектовање складишта података

На основу софтверских класа структуре пројектују се табеле (складишта података) релационог система за управљање базом података.

Табела User:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
username	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
password	VARCHAR(200)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
first_name	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
last_name	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
email	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gender	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
birth_date	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
phone_number	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
favourite_club_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 19. Складиште података: Табела User

Табела Role:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 20. Складиште података: Табела Role

Табела UserRole:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
user_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
role_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 21. Складиште података: Табела UserRole

Табела Club:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
url	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
image	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
manager	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 22. Складиште података: Табела Club

Табела Player:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
club_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
url	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(40)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
nationality	VARCHAR(40)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
birth_date	VARCHAR(40)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
age	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
position	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
height	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
weight	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
image	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
price	DECIMAL(3,1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
total_points	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 23. Складишице података: Табела Player

Табела Team:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
user_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
total_points	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
free_transfers	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
captain_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
vice_captain_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
money_remaining	DECIMAL(3,1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 24. Складишице података: Табела Team

Табела TeamPlayer:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
team_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
points	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
on_bench	TINYINT(1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				

Табела 25. Складишице података: Табела TeamPlayer

Табела League:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
owner_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 26. Складиште података: Табела League

Табела TeamLeagueMembership:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
team_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
league_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 27. Складиште података: Табела TeamLeagueMembership

Табела Gameweek:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
order_number	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 28. Складиште података: Табела Gameweek

Табела TeamGameweekPerformance:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
team_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweek_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
points	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 29. Складиште података: Табела TeamGameweekPerformance

Табела PlayerGameweekPerformance:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
player_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweek_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
points	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 30. Складиште података: Табела PlayerGameweekPerformance

Табела Match:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
gameweek_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
url	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
result	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
date_time	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
venue	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
host_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
guest_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
sent	TINYINT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Табела 31. Складиште података: Табела Match

Табела MatchEvent:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
match_id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweekId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
minute	VARCHAR(10)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
result	VARCHAR(10)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
club_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 32. Складиште података: Табела MatchEvent

Табела Goal:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
matchEventId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
matchId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweekId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
goal_player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
own_goal	TINYINT(1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 33. Складиште података: Табела Goal

Табела Card:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
matchId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweekid	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
card	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 34. Складиште података: Табела Card

Табела MinutesPlayed:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
matchEventId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
matchId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweekId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
minutes_played	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 35. Складиште података: Табела MinutesPlayed

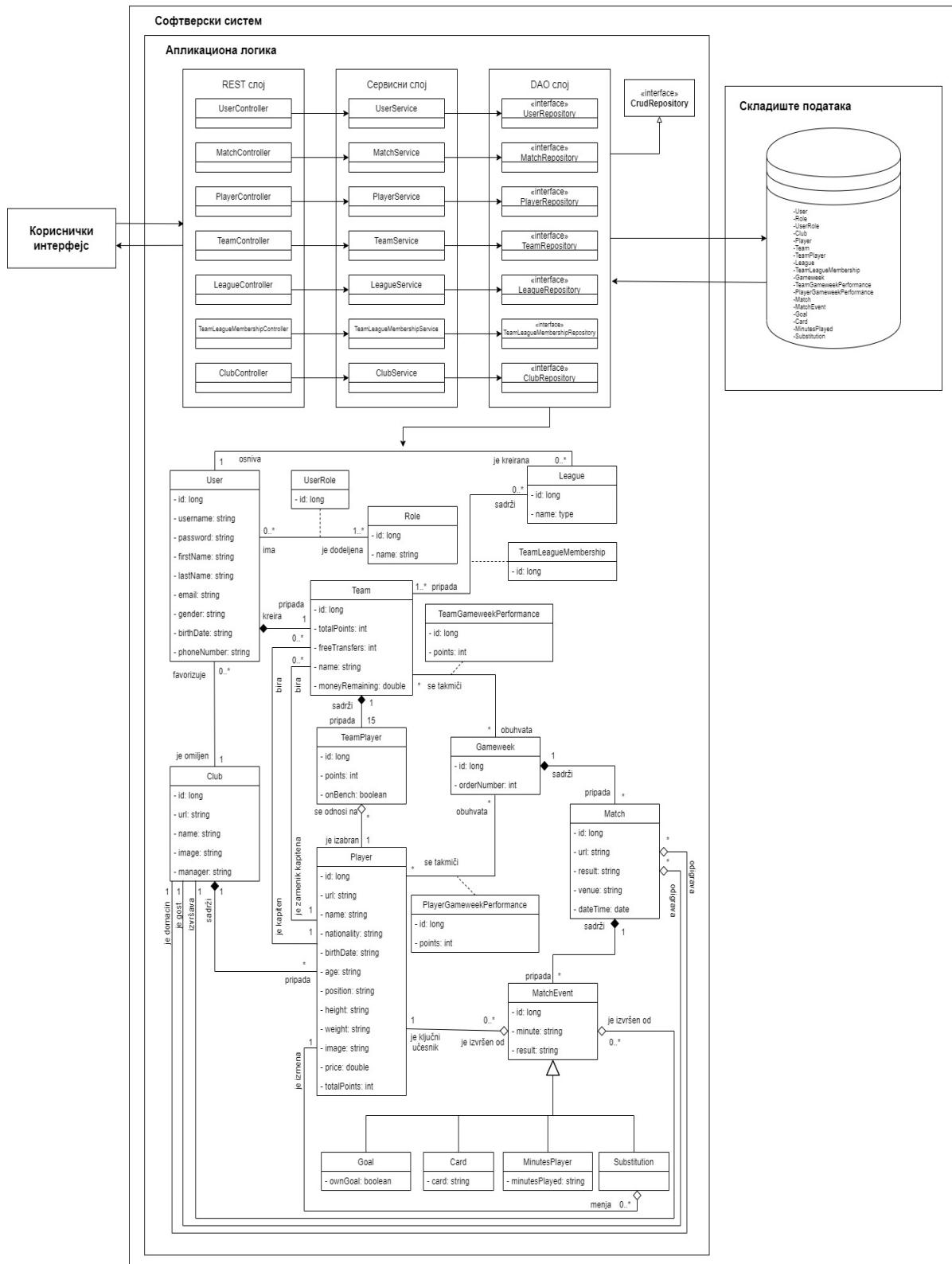
Табела Substitution:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
matchEventId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
matchId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
gameweekId	BIGINT(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
in_player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
out_player_id	BIGINT(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
created_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
modified_on	TIMESTAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Табела 36. Складиште података: Табела Substitution

4.3.5. Коначан изглед архитектуре софтверског система

Коначан изглед архитектуре софтверског система након фазе пројектовања приказан је на слици 75:



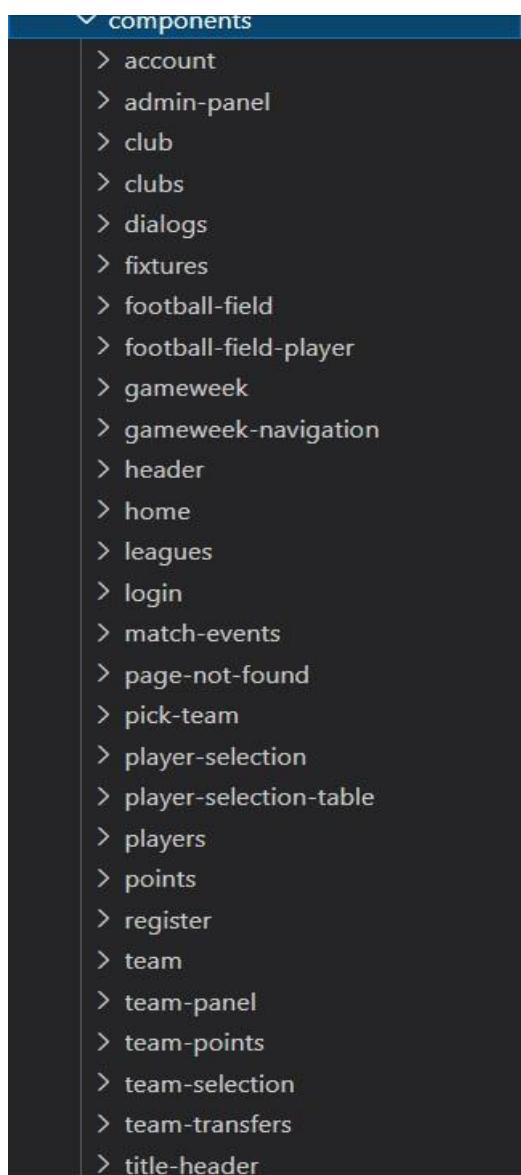
Слика 75. Коначан изглед архитектуре софтверског система.

4.4. Имплементација

Софтверски систем је развијен као клијент сервер апликација. Серверске апликације, обе архитектуре, су развијене у Јава ЕЕ окружењу, употребом Spring Boot радног оквира унутар развојног окружења Eclipse IDE. За потребе развоја микросервиса је коришћен радни оквир Spring Cloud. Систем за управљање базом података који је коришћен у развоју је MySQL. Као алат за објектно-релационо мапирање и перзистенцију података коришћен је Spring Data JPA. За управљање верзијама базе података је коришћен алат Liquibase.

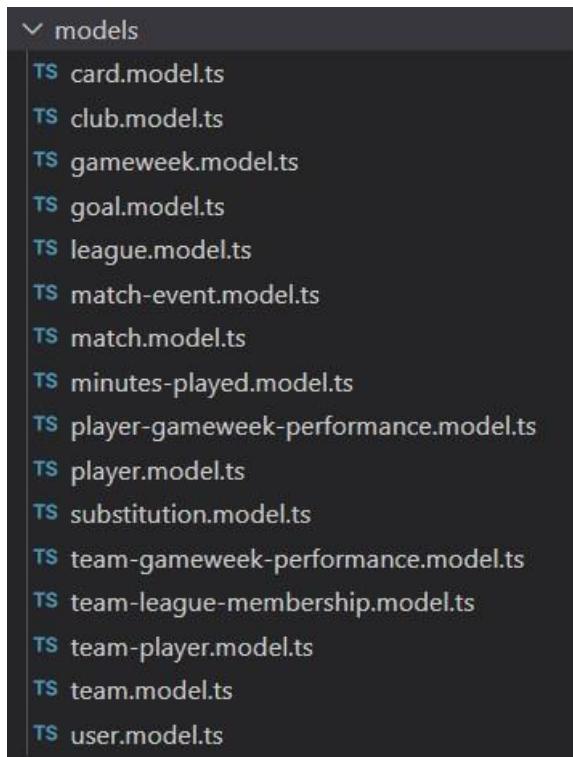
За развој клијентске стране коришћен је радни оквир Angular. За комуникацију између клијентске и серверске стране коришћен је REST API.

У наставку је приказана структура клијентске стране. Компоненте клијентске апликације су приказане на слици 76.



Слика 76. Компоненте клијентске апликације

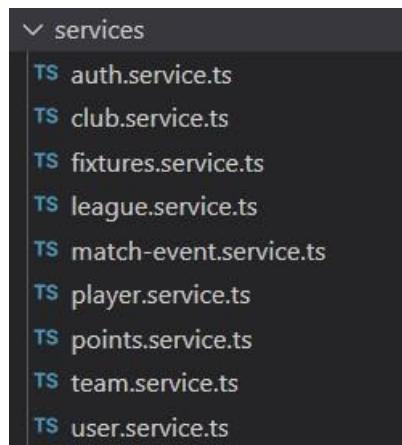
На слици 77 су приказани модели клијентске апликације.



The screenshot shows a dark-themed file explorer window. A folder named 'models' is expanded, revealing its contents. Each item is preceded by a small blue 'TS' icon, indicating they are TypeScript files. The listed files are: card.model.ts, club.model.ts, gameweek.model.ts, goal.model.ts, league.model.ts, match-event.model.ts, match.model.ts, minutes-played.model.ts, player-gameweek-performance.model.ts, player.model.ts, substitution.model.ts, team-gameweek-performance.model.ts, team-league-membership.model.ts, team-player.model.ts, team.model.ts, and user.model.ts.

Слика 77. Модели клијентске апликације

На слици 78 су приказани сервиси клијентске апликације.



The screenshot shows a dark-themed file explorer window. A folder named 'services' is expanded, revealing its contents. Each item is preceded by a small blue 'TS' icon. The listed files are: auth.service.ts, club.service.ts, fixtures.service.ts, league.service.ts, match-event.service.ts, player.service.ts, points.service.ts, team.service.ts, and user.service.ts.

Слика 78. Сервиси клијентске апликације

У наставку је приказана структура серверске стране. Класе пакета *domain* су приказане на слици 79.



Слика 79. Пакет *domain*

Класе пакета *repository* су приказане на слици 80.



Слика 80. Пакет *repository*

Класе пакета *service* су приказане на слици 81.



Слика 81. Пакет *service*

Класе пакета *rest* су приказане на слици 82.



Слика 82. Пакет *rest*

4.5. Тестирање

Сваки од наведених и имплементираних случајева коришћења је тестиран. Приликом тестирања уношени су и правилни и неправилни подаци како би се утврдиле грешке или неусаглашености и како би оне биле исправљене. Након тестирања софтверски систем је спреман за рад.

За тестирање је коришћен JUnit радни оквир. Написани су unit тестови приказани на слици 83.



Слика 83. Unit тестови

5. Упоредна анализа примене монолитне и микросервисне архитектуре у развоју софтверских система

У оквиру овог одељка је на практичном примеру извршена упоредна анализа употребе монолитне и микросервисне архитектуре у развоју софтверских система. На самом почетку је дата формулатија проблема са предлогом решења. Након тога је прецизно описан поступак мерења резултата, наведени су критеријуми поређења, метрике и поступци коришћени у истраживању. На крају су представљени добијени резултати анализе заједно са визуелизацијама и донесеним закључцима.

5.1. Формулација проблема

Упоредни преглед монолитних и микросервисних апликација, који представља главну проблематику овог рада, потребно је употребити анализом њихових перформанси на конкретном примеру. Са тим у вези, потребно је на истом студијском примеру развићи апликације применом наведених архитектуралних стилова, тестирати њихово понашање под унапред дефинисаним, симулираним условима и забележити резултате тестирања чијом ће се анализом на крају доћи до конкретних закључака.

Све је више присутна тенденција великих компанија да развијају сопствене дистрибуције Јава виртуелне машине. Промена политике лиценцирања компаније Oracle, по којој Јава није више бесплатна за комерцијалну употребу, узроковала је окретање корисника ка алтернативним решењима. Истраживање компаније Snyk из 2021. године показује да се у производном окружењу највише користи AdoptOpenJDK у проценту од 44.1%.^[43] Oracle JDK се налази на трећем месту што је показатељ да је и даље веома значајан део Јава екосистема. Иза ње се налазе: Azul Zulu, Amazon Corretto, GraalVM, Alibaba Dragonwell и друге дистрибуције.^[43] Са тим у вези, потребно је да у оквиру рада предмет анализе буду и перформансе монолитних и микросервисних апликација у раду са GraalVM виртуелном машином. Конкретно, потребно је тестирати перформансе GraalVM native image технологије чија је главна одлика компајлирање пре времена извршења (Ahead Of Time).

Циљ анализе је добити одговоре на следећа питања:

- Која је разлика у перформансама монолитних и микросервисних апликација?
- Како миграција монолита у микросервисе утиче на перформансе апликације?
- У којим ситуацијама је пожељно применити монолитну архитектуру?
- У којим ситуацијама је пожељно применити микросервисну архитектуру?
- Какав је утицај примене GraalVM native image (AOT) технологије на перформансе монолитних и микросервисних апликација у поређењу са стандардном Јава виртуелном машином (JIT)?

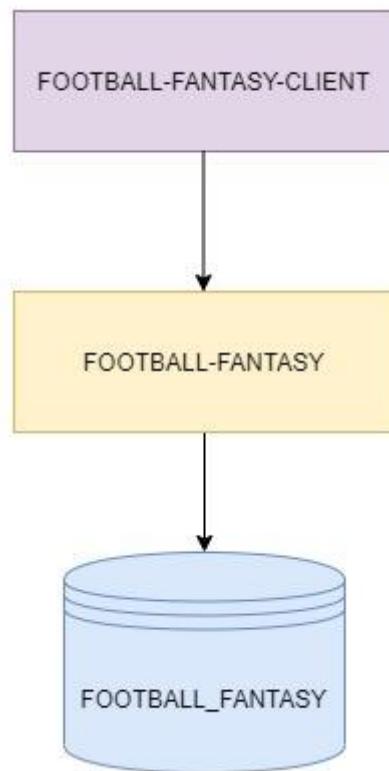
5.2. Предлог решења

Како би упоредна анализа монолитне и микросервисне архитектуре била могућа, за потребе тестирања ће бити развијене апликације на истом студијском примеру Football Fantasy игре. Она представља адекватан пример јер је довољно комплексна за развој применом монолитне, као и микросервисне архитектуре. У циљу веродостојне процене утиција миграције монолитног система у микросервисни на перформанс, првенствено ће бити развијена монолитна апликација. Она ће затим, инкременталним приступом, бити декомпонована на микросервисе комбинованом применом следеће две стратегије:

- Декомпозиција по функционалности
- Декомпозиција по доменском објекту

Клијентска апликација ће бити развијена применом радног оквира Angular. Она није предмет анализе и јединствена за све случајеве. На серверској страни ће на истом студијском примеру бити развијене апликације у четири верзије чији је детаљан опис дат у наставку:

1. Монолитна апликација (JVM) - Монолитна Spring Boot апликација са јединственом MySQL базом података. Изглед архитектуре ове верзије је приказан на слици 84.

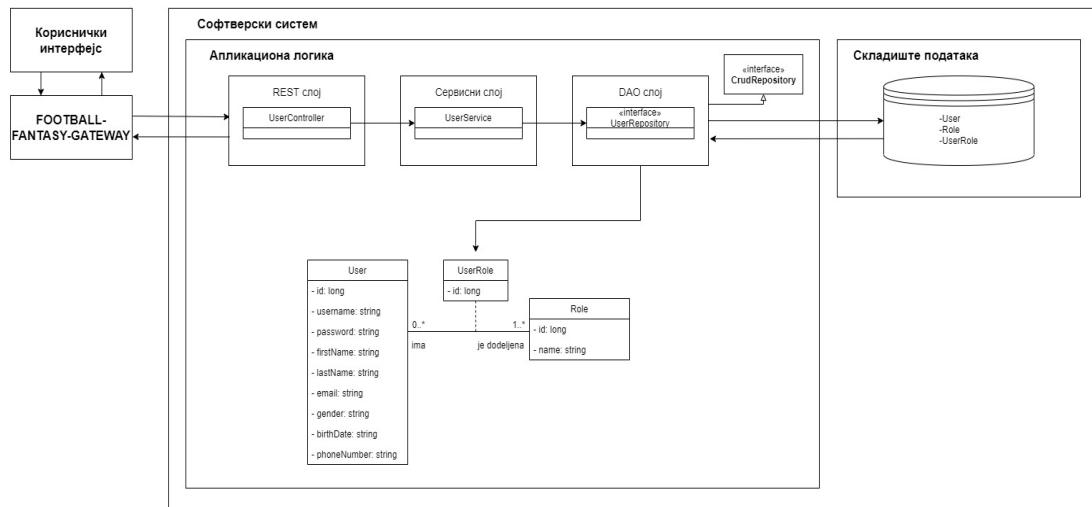


Слика 84. Архитектура монолитне апликације

2. Микросервисна апликација (JVM) – Као резултат инкременталног декомпоновања монолитне апликације применом стратегија декомпозиције по функцији и доменском објекту настаће следећи микросрвиси:

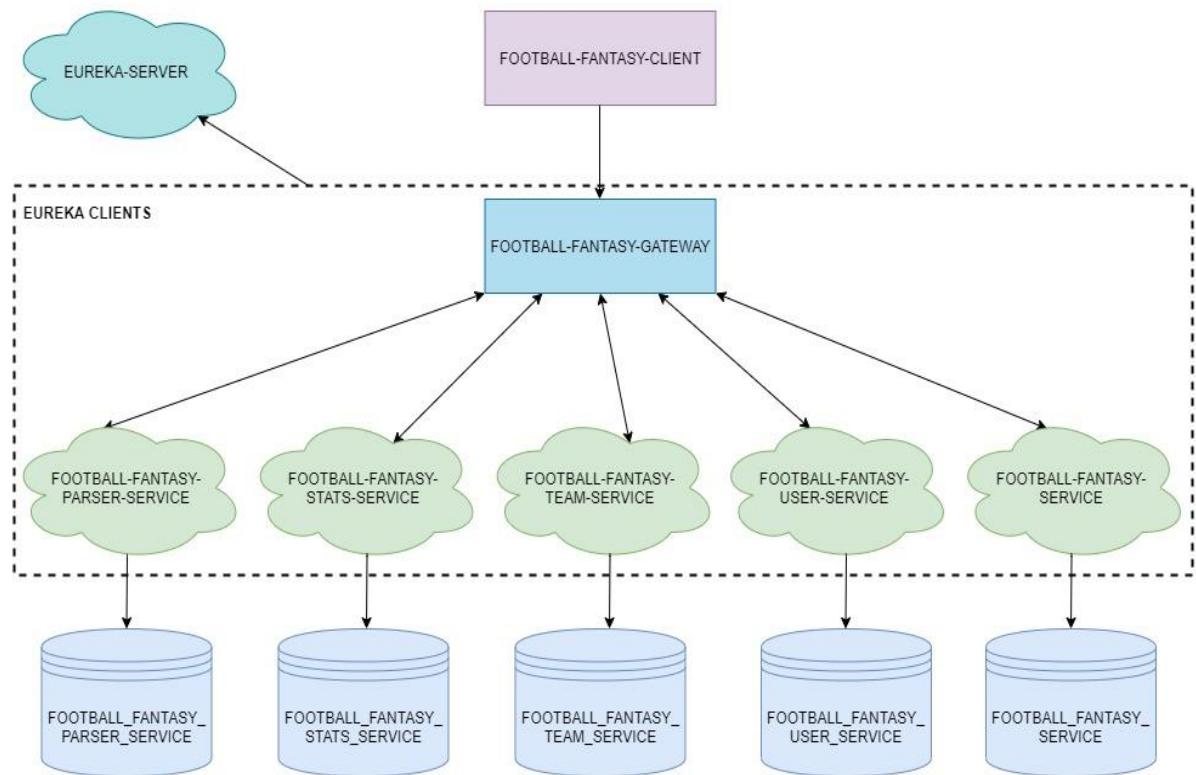
- football-fantasy-parser-service - функција парсирања података о мечевима, тимовима и играчима са екстерних извора
- football-fantasy-stats-service – функција рачунања статистика и поена
- football-fantasy-team-service – управљање доменским објектом Team
- football-fantasy-user-service – управљање доменским објектом User
- football-fantasy-service – остатак монолита за који се није показало оправданим даље декомпоновање јер садржи често коришћене функције од стране других микросрвиса којима не припадају логички
- football-fantasy-gateway – прихватање клијентских захтева и даље прослеђивање одговарајућем сервису

Коначан изглед архитектуре монолитног софтверског система је приказан раније на слици 75. Сваки од микросрвиса је добијен копирањем одговарајућих делова програмског кода монолита те је њихова архитектура потпуно иста. На слици 85 је приказан коначан изглед архитектуре football-fantasy-user-service микросрвиса, која је на исти начин примењена и за све остале микросрвисе.



Слика 85. Коначан изглед архитектуре микросрвиса football-fantasy-user-service

За имплементацију ће се додатно користити радни оквир Spring Cloud и његови модули. Улогу централног регистра микросрвиса имаће Spring Cloud Netflix Eureka, док ће Gateway API бити имплементиран применом Spring Cloud Gateway модула. Spring Cloud OpenFeign ће се користити за синхрону комуникацију између микросрвиса. Сваки микросрвис ће поседовати сопствену MySQL базу података. Изглед архитектуре микросрвисне апликације је приказан на слици 86.



Слика 86. Архитектура микросрвисне апликације

3. Монолитна апликација (GraalVM) – Апликација имплементирана на исти начин као апликација под редним бројем један, са разликом да се за њено компајлирање и покретање користи GraalVM применом пројекта Spring Native.
4. Микросрвисна апликација (GraalVM) – Апликација имплементирана на исти начин као апликација под редним бројем два, са разликом да се за њено компајлирање и покретање користи GraalVM применом пројекта Spring Native.

Након имплементације ће бити извршено међусобно поређење свих наведених верзија апликације анализом њихових перформанси под контролисаним условима. Тестираће се њихово понашање под различитим оптерећењима, симулацијом интеракције корисника са системом путем унапред дефинисаних тест планова. Такође, биће развијен сопствени механизам за бележење, прикупљање, визуелизацију и анализу резултата тестирања. На крају ће се из добијених резултата извући одговарајући закључци и препоруке.

5.3. Мерење резултата

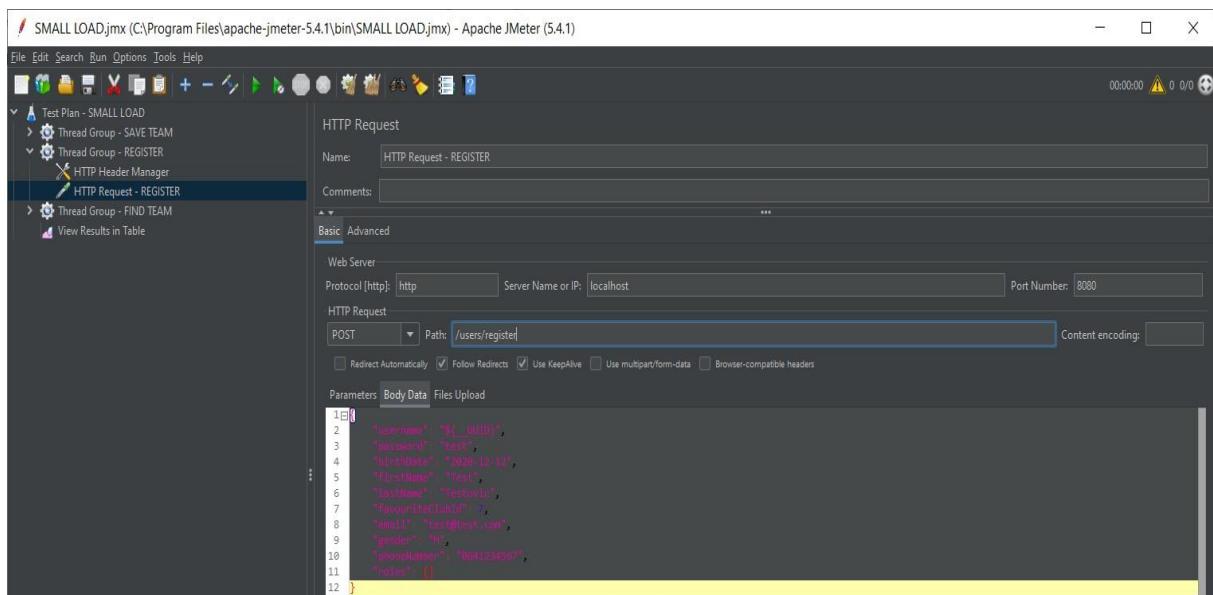
За потребе симулације интеракције корисника са системом је коришћен алат JMeter. Перформансе сваке појединачне апликације су тестиране под различитим оптерећењима, дефинисаним у оквиру тест планова следећих подешавања:

1. Оптерећење – 100 корисника
2. Оптерећење – 1000 корисника
3. Оптерећење – 10000 корисника

Може се приметити да је свако наредно оптерећење десет пута веће у односу на претходно. Број захтева се повећава постепено. Време које је потребно да се активира максимални дефинисани број корисника сваког тест плана у секундама износи 10% од укупног броја корисника. То значи да ће при оптерећењу од 100 корисника бити потребно укупно 10 секунди како би се сви покренули, додавањем 10 корисника по секунди. Корисници у оквиру плана су распоређени у три групе по REST захтевима које истовремено шаљу ка систему. За потребе тестирања су изабрани случајеви коришћења који се најчесталије користе у пракси:

1. Креирање тима (POST) – 30% укупног броја захтева
2. Регистрација корисника (POST) – 30% укупног броја захтева
3. Преглед тима (GET) – 40% укупног броја захтева

Највећи проценат захтева је дефинисан за операцију прегледа тима јер је по претпоставци то најчешћи случај коришћења, док се регистрација и креирање тима позивају у истом односу па им је одређен једнак проценат захтева. Пример конфигурисаног JMeter тест плана је дат на слици 87.



Слика 87. JMeter тест план

Конкретне метрике чије су вредности анализиране за потребе поређења перформанси свих апликација су:

1. Време извршења кључних метода – време потребно за извршење претходно наведених кључних метода апликације (креирање тима, регистрација корисника и преглед тима) у милисекундама
2. Кашњење (енгл. Latency) – време протекло између тренутка слања захтева и тренутка пријема првог одговора у милисекундама [44]
3. Проценат успешности – број успешно извршених захтева у односу на укупан број захтева изражен у процентима
4. Време изградње извршног програма (енгл. Build time) – време потребно за изградњу извршног програма у секундама
5. Време покретања апликације (енгл. Startup time) – време покретања апликације у стање доступно кориснику у секундама
6. Употреба heap меморије – количина коришћене heap меморије у бајтовима
7. Употреба процесора – искоришћеност ресурса процесора изражена у процентима

Изабране метрике су подржане од стране коришћених алата: JMeter, Metricbeat и сопствене Spring Boot библиотеке за мониторинг. [44, 45] Вредности наведених метрика су мерење на различите начине. За потребе мерења **времена извршења кључних метода и употребе heap меморије** је развијена посебна библиотека за мониторинг базирана на аспектно-оријентисаном програмирању (AOP), која је затим као зависност укључена у све апликације. Она обухвата `@Monitoring` анотацију и аспект за уписивање кључних кључних података у лог фајл. При извршавању сваке методе означене овом анотацијом у лог фајл се уписују следећи подаци:

- Виртуелна машина – JVM, GRAALVM
- Архитектура - MONOLITHIC, MICROSERVICE
- Оптерећење - 100, 1000, 10000
- Време извршења методе
- Употреба heap меморије

У наставку је дат пример генерисаног лога у JSON формату:

```
{  
    "MEMORY":{  
        "USED_NON_HEAP_MEMORY":121916768,  
        "FREE_MEMORY":260888192,  
        "USED_MEMORY":243476864,  
        "USED_HEAP_MEMORY":243476864,  
        "TOTAL_MEMORY":504365056  
    },  
    "LOAD":100,  
    "ARCHITECTURE":"MONOLITHIC",  
    "METHOD":"findByIdSmall",  
    "VIRTUAL_MACHINE":"JVM",  
    "TIME":"17.07.2021. 19:55:49:744",  
    "CLASS":"com.fon.footballfantasy.rest.TeamController",  
    "DURATION":323  
}
```

Начин исправног означавања методе `@Monitoring` анотацијом се може видети на следећем примеру:

```
@Monitoring(architecture = MONOLITHIC, virtualMachine = JVM, load = 100)  
@PostMapping(value = "/team")  
ResponseEntity<?> save(@RequestBody Team team) {  
    return new ResponseEntity<>(teamService.save(team), HttpStatus.OK);  
}
```

Кашњење и проценат успешности су мерени помоћу алата JMeter и CSV формату.

Време изградње извршног програма, које укључује процесе као што су компајлирање, извршавање тестова, паковање и други, и **време покретања апликације** мерени су ручно и уписивани у CSV фајл. Ове метрике су мерене тако што је сваки процес покренут по три пута, након чега је израчуната средња вредност измерених резултата.

Употреба процесора је мерена помоћу *Metricbeat* агента.

Генерисане податке даље преузима *Elastic Stack* који чине следеће технологије: *Filebeat*, *Metricbeat*, *Logstash*, *Elasticsearch* и *Kibana*. *Filebeat* и *Logstash* имају улогу агената који прате стање лог фајлова и сваки нови лог складиште у *Elasticsearch* бази података. *Metricbeat* је агент чија је улога бележење системских метрика које се такође прослеђују ка *Elasticsearch* бази. *Kibana* као презентациони слој *stack-a*, извршава упите над *Elasticsearch* базом како би креирала појединачне визуелизације, као и комплетан *dashboard* са агрегираним резултатима тестирања који ће бити приказани у наредним

поглављима. Преостали CSV фајлови са резултатима мерења су ручно увезени у Kibani за потребе анализе.

Сви тестови су извршени на машини са следећим хардверским спецификацијама:

- Processor: Intel(R) Core(TM) i7-8750 CPU @ 2.20GHz 2.21GHz
- RAM: 16 GB
- GPU: GeForce® GTX 1050Ti
- SSD: 512 GB
- OS: Windows 10 Pro

5.4. Резултат анализе

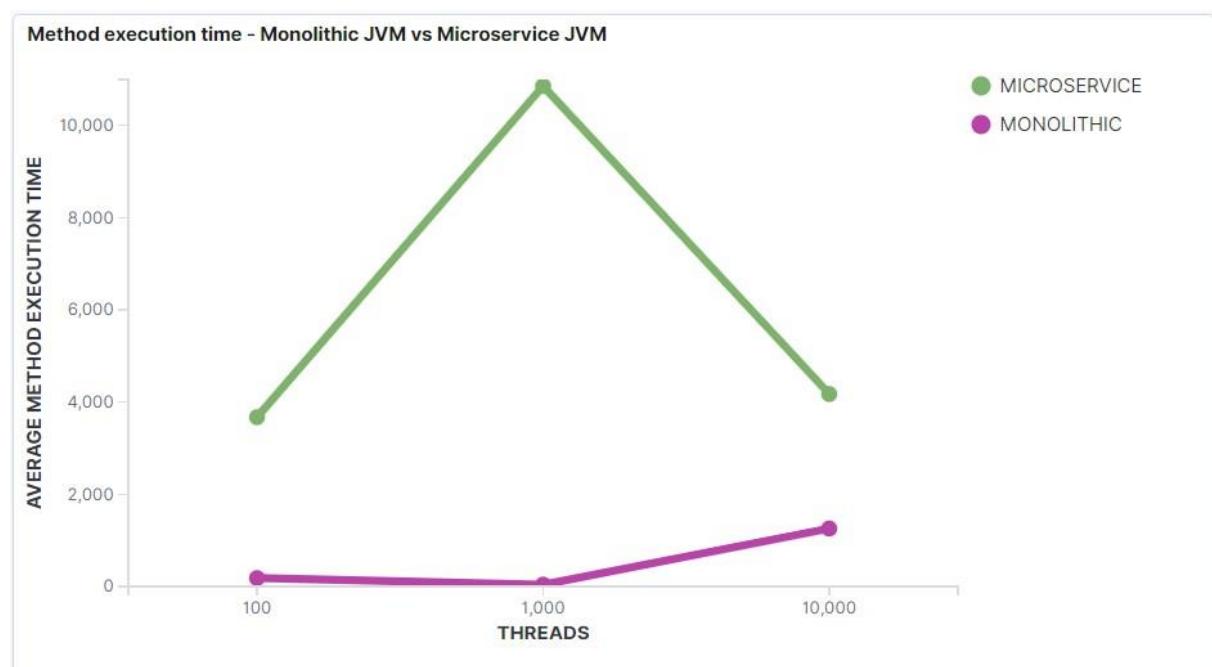
У оквиру овог поглавља су представљени добијени резултати тестирања.

5.4.1. Поређење монолитне (JVM) и микросервисне (JVM) апликације

У оквиру овог поглавља је извршена упоредна анализа добијених резултата монолитне (JVM) и микросервисне (JVM) апликације.

5.4.1.1. Време извршења кључних метода

На слици 88 је приказано време извршења кључних метода монолитне и микросервисне апликације на Јава виртуелној машини.

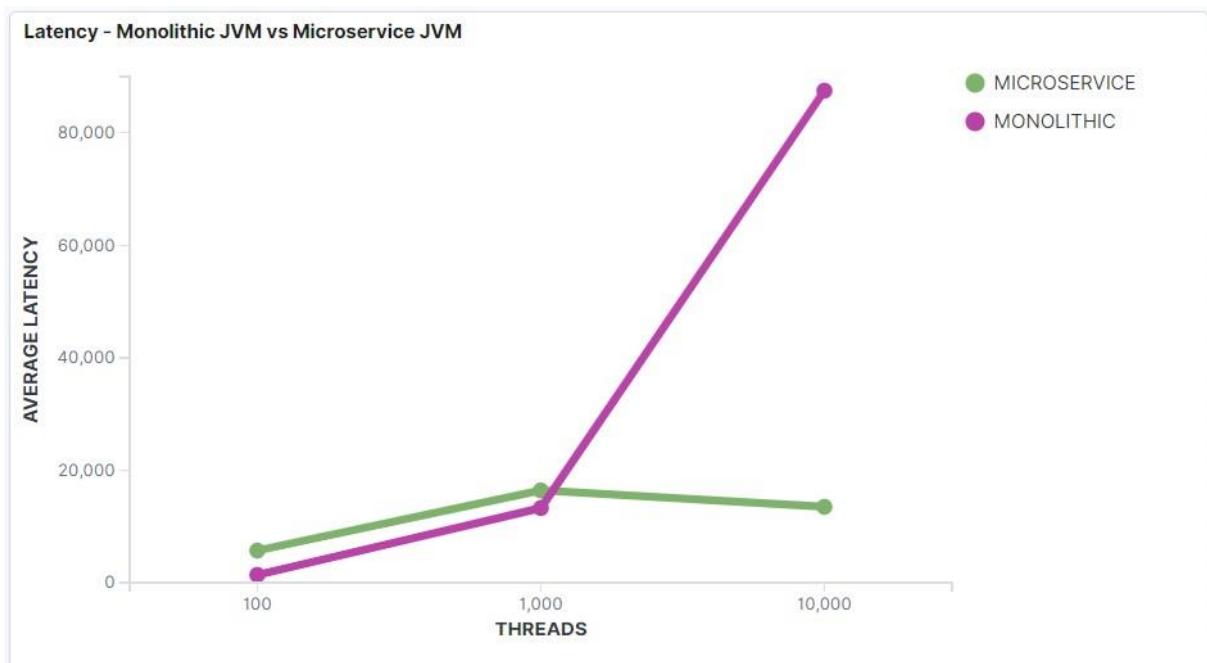


Слика 88. Време извршења - монолитна (JVM) и микросервисна (JVM) апликација

Монолитна апликација је, под свим тестираним количинама оптерећења, показала боље резултате од микросервисне апликације, вишеструко већом брзином извршења метода. При оптерећењу од 100, 1000 и 10000 корисничких захтева монолитна апликација је извршавала кључне методе са просечним временом од 181.76, 39.264 и 1,257.508 милисекунди респективно. Са друге стране, микросервисној апликацији је у просеку било потребно 3,673.629, 10,858.063 и 4,175.564 милисекунди. Највећа разлика је постигнута при оптерећењу од хиљаду захтева, где је просечна брзина већа приближно 278 пута у корист монолитне апликације.

5.4.1.2. Кашњење

На слици 89 је приказано измерено кашњење монолитне и микросрвисне апликације на Јава виртуелној машини.



Слика 89. Кашњење - монолитна (JVM) и микросрвисна (JVM) апликација

При оптерећењу од 100 и 1000 захтева, боље резултате је показала монолитна апликација са просечним кашњењем од 1,381.74 и 13,275.424 милисекунди респективно, у поређењу са 5,705.89 и 16,401.315 милисекунди које је забележила микросрвисна апликација. Може се приметити да разлика није значајна јер су измерени резултати приближни. Међутим, при оптерећењу од 10000 захтева, микросрвисна апликација показује приближно 6.5 пута бољи резултат са просечним кашњењем 13,490.625 милисекунди у односу на 87,499.284 милисекунди које бележи микросрвис.

5.4.1.3. Проценат успешности

На слици 90 је приказан измерени проценат успешности монолитне апликације на Јава виртуелној машини.



Слика 90. Проценат успешности - монолитна (JVM) апликација

При оптерећењу од 100 и 1000 захтева, монолитна апликација је забележила успешност од 100%, док је за 10000 захтева забележена успешност од 57.61%.

На слици 91 је приказан измерени проценат успешности микросервисне апликације на Јава виртуелној машини.

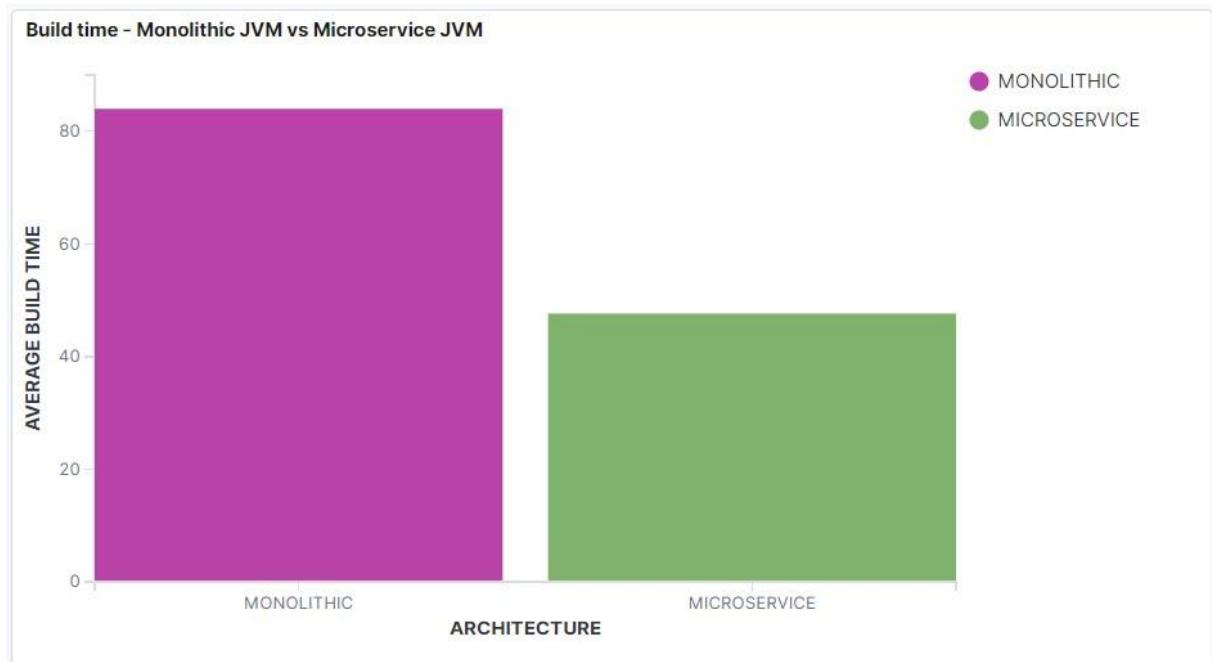


Слика 91. Проценат успешности - микросервисна (JVM) апликација

Са друге стране, микросервисна апликације је при оптерећењу од 100 и 1000 захтева забележила исти резултат, док је за 10000 захтева успешност мања у односу на монолитну са веома лошим резултатом од 37.39%.

5.4.1.4. Време изградње извршног програма

На слици 92 је приказано време изградње извршног програма монолитне и микросервисне апликације на Јава виртуелној машини.

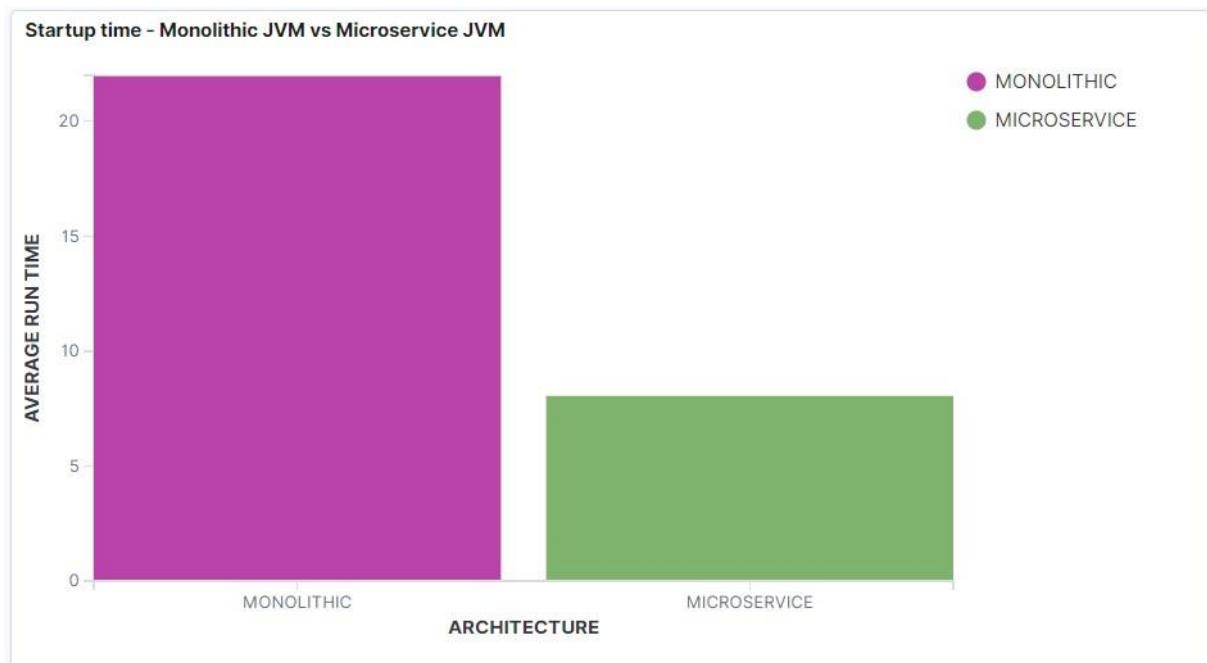


Слика 92. Време изградње извршног програма - монолитна (JVM) и микросервисна (JVM) апликација

За изградњу извршног програма микросервисне апликације је у просеку потребно 47.641 секунди у односу на 84 секунде које бележи монолит, што је приближно 1.7 пута бољи резултат у корист микросервиса.

5.4.1.5. Време покретања апликације

На слици 93 је приказано време покретања монолитне и микросервисне апликације на Јава виртуелној машини.

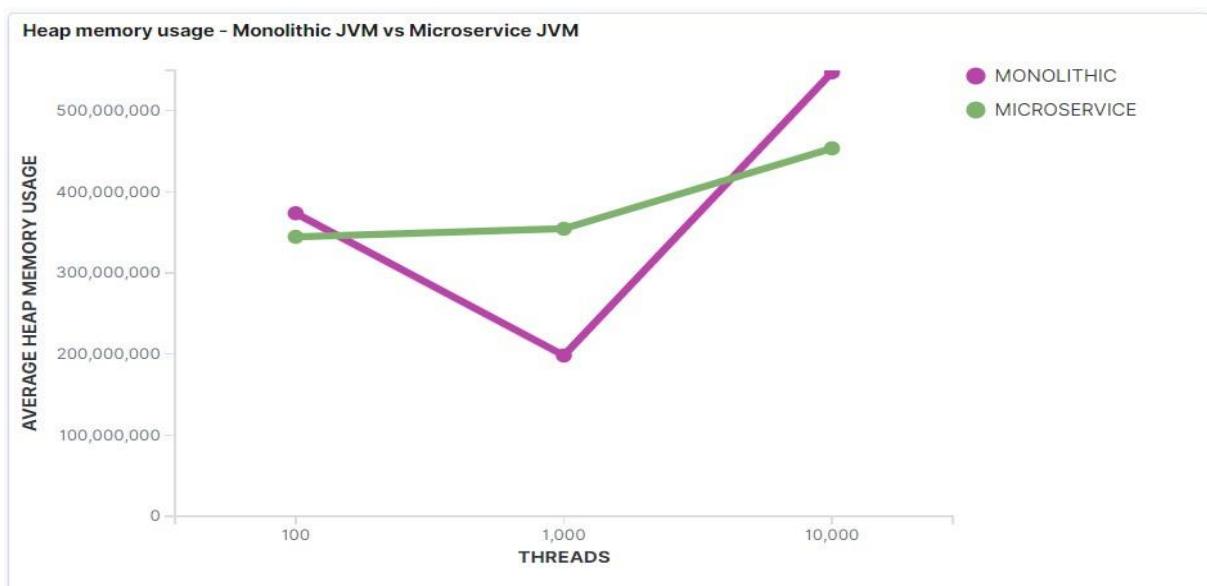


Слика 93. Време покретања апликације - монолитна (JVM) и микросервисна (JVM) апликација

За покретање микросервисне апликације је у просеку потребно 8.053 секунди у односу на 21.967 секунде које бележи монолит, што је приближно 2.7 пута бољи резултат у корист микросервиса.

5.4.1.6. Употреба heap меморије

На слици 94 је приказана употреба heap меморије монолитне и микросервисне апликације на Јава виртуелној машини.

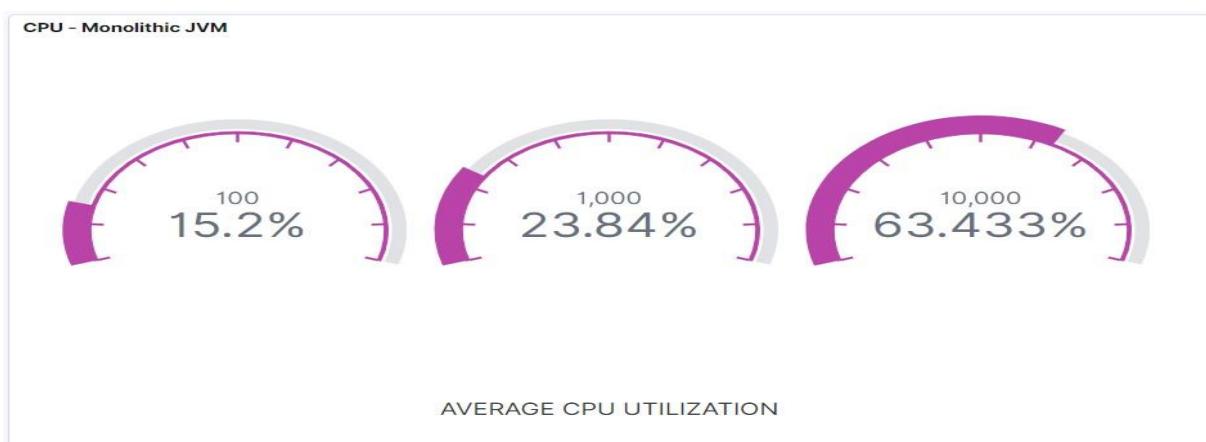


Слика 94. Употреба heap меморије - монолитна (JVM) и микросервисна (JVM) апликација

При оптерећењу од 100 и 10000 захтева, боље резултате показује микросервисна апликација са просечном употребом heap меморије од 344,660,001.412 и 453,854,881.578 байтова респективно, у поређењу са монолитом који у просеку бележи 373,775,873.413 и 547,677,220.182 байтова употребљене меморије. Највећа разлика се примећује при оптерећењу од 1000 захтева где монолит користи приближно 1.8 пута мање heap меморије од микросрвиса.

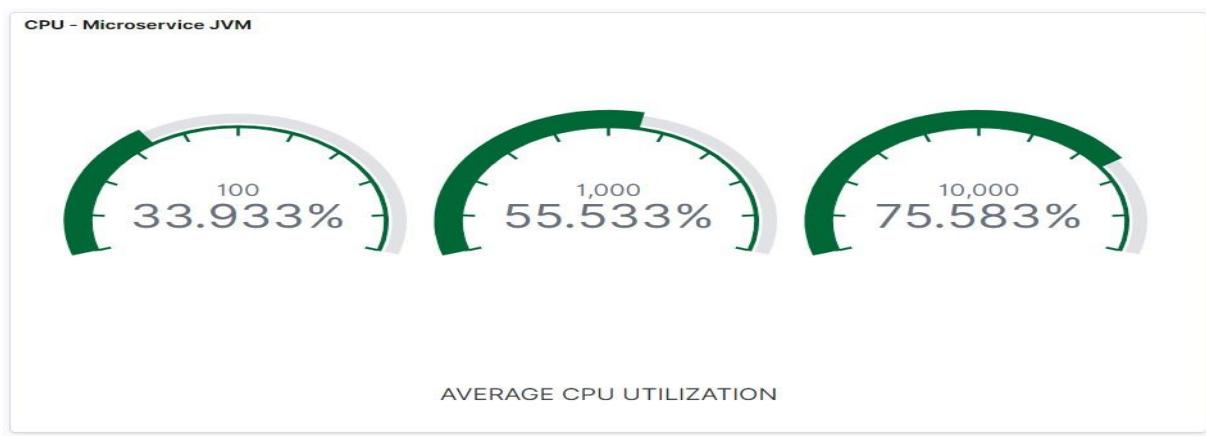
5.4.1.7. Употреба процесора

На слици 95 је приказана употреба ресурса процесора монолитне апликације на Јава виртуелној машини.



Слика 95. Употреба процесора - монолитна (JVM) апликација

Монолитна апликација, за оптерећења од 100, 1000 и 10000 захтева, бележи употребу процесора од 15.2%, 23.84% и 63.433% респективно. На слици 96 је приказана употреба ресурса процесора микросрвисне апликације на Јава виртуелној машини.



Слика 96. Употреба процесора - микросрвисна (JVM) апликација

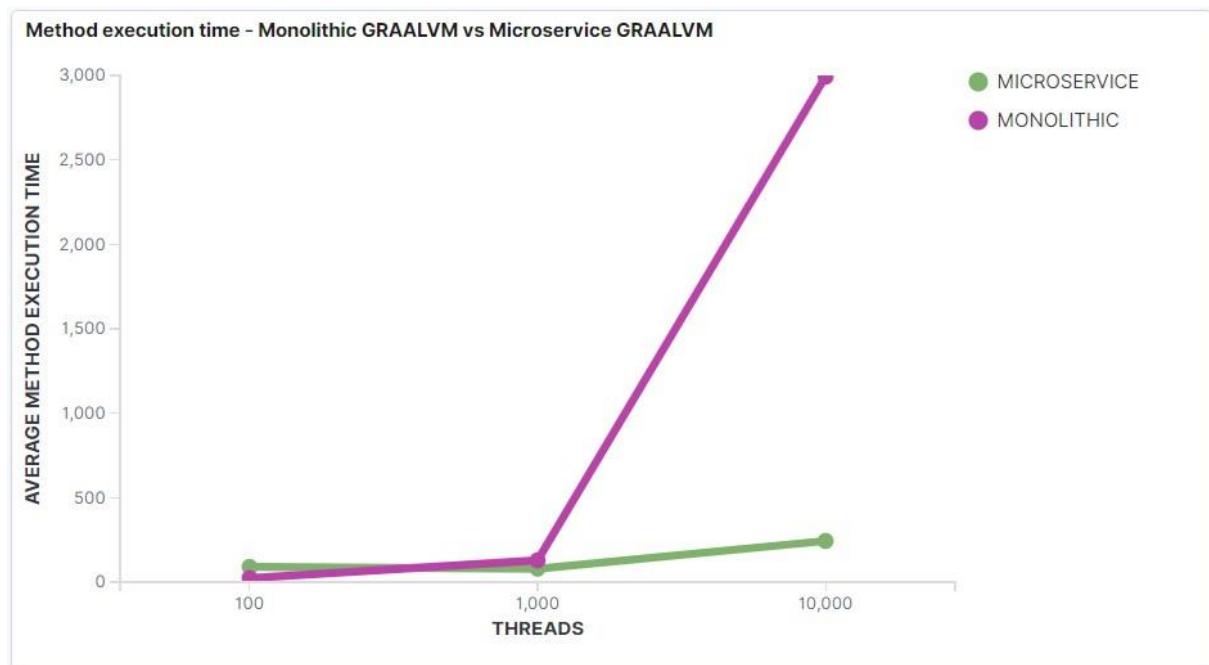
Са друге стране, микросрвисна апликација захтева већу количину ресурса процесора, за све наведене количине захтева, са забележеним резултатима од 33.933%, 55.533% и 75.583% респективно.

5.4.2. Поређење монолитне (GraalVM) и микросрвисне (GraalVM) апликације

У оквиру овог поглавља је извршена упоредна анализа добијених резултата монолитне (GraalVM) и микросрвисне (GraalVM) апликације.

5.4.2.1. Време извршења кључних метода

На слици 97 је приказано време извршења кључних метода монолитне и микросрвисне апликације на GraalVM виртуелној машини.

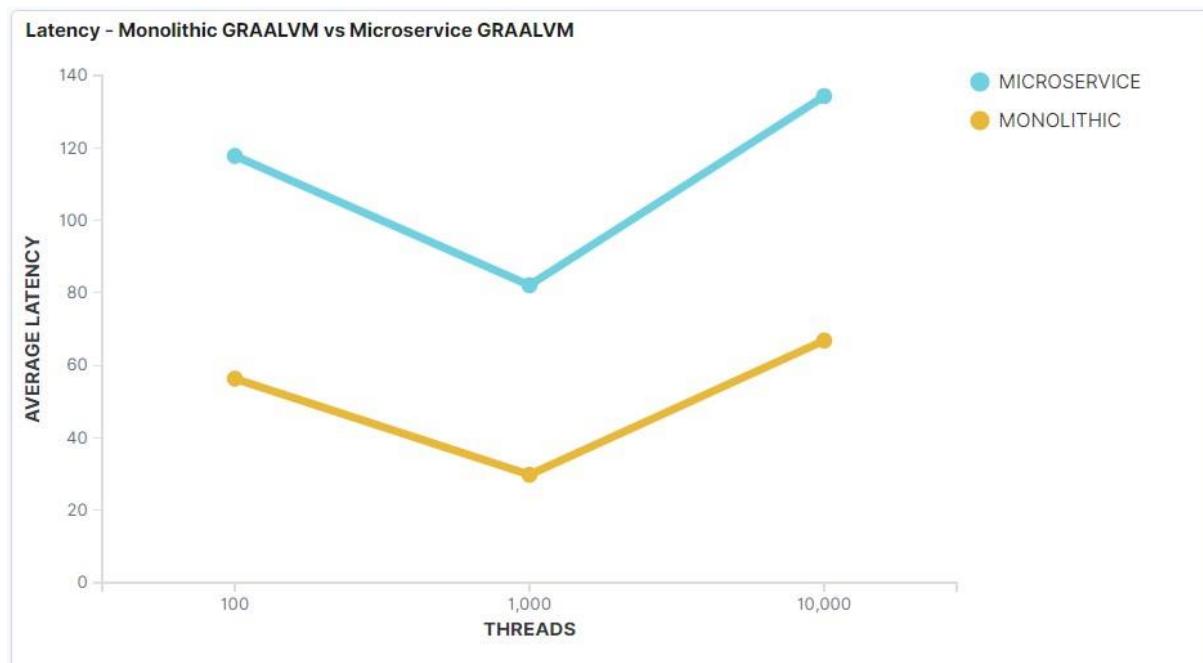


Слика 97. Време извршења - монолитна (GraalVM) и микросрвисна (GraalVM) апликација

При оптерећењу од 100 и 1000 захтева, монолита и микросрвисна апликација су показале приближне резултате. Велика предност у корист микросрвисне апликације је забележена при оптерећењу од 10000 корисничких захтева са приближно 12.3 пута већом просечном брзином извршења кључних метода. Прецизније, за наведена оптерећења, монолитна апликација је забележила време извршења метода у просеку од 24.27, 128.549 и 2,992.59 милисекунди респективно, за разлику од микросрвисне чији су резултати 89.89, 79.59 и 243.673 милисекунди.

5.4.2.2. Кашњење

На слици 98 је приказано измерено кашњење монолитне и микросрвисне апликације на GraalVM виртуелној машини.



Слика 98. Кашњење - монолитна (GraalVM) и микросрвисна (GraalVM) апликација

Монолитна апликација је забележила боље резултате са 2 до 3 пута мањим просечним кашњењем у поређењу са микросрвисном апликацијом, независно од количине оптерећења. За 100, 1000 и 10000 захтева, просечно кашњење монолита износи 56.31, 29.797 и 66.852 милисекунди респективно, у односу на микросрвисе који бележе резултате од 117.8, 82.115 и 134.303 милисекунди. Може се приметити да, у погледу овог критеријума, обе апликације најбоље подносе средњу количину оптерећења.

5.4.2.3. Проценат успешности

Обе апликације су показале исте или приближне резултате, независно од количине оптерећења. На слици 99 је приказан измерени проценат успешности монолитне апликације на GraalVM виртуелној машини.



Слика 99. Проценат успешности - монолитна (GraalVM) апликација

При оптерећењу од 100 и 1000 захтева, монолитна апликација је забележила успешност од 100%, док је за 10000 захтева забележена успешност од 58.2%. На слици 100 је приказан измерени проценат успешности микросервисне апликације на GraalVM виртуелној машини.

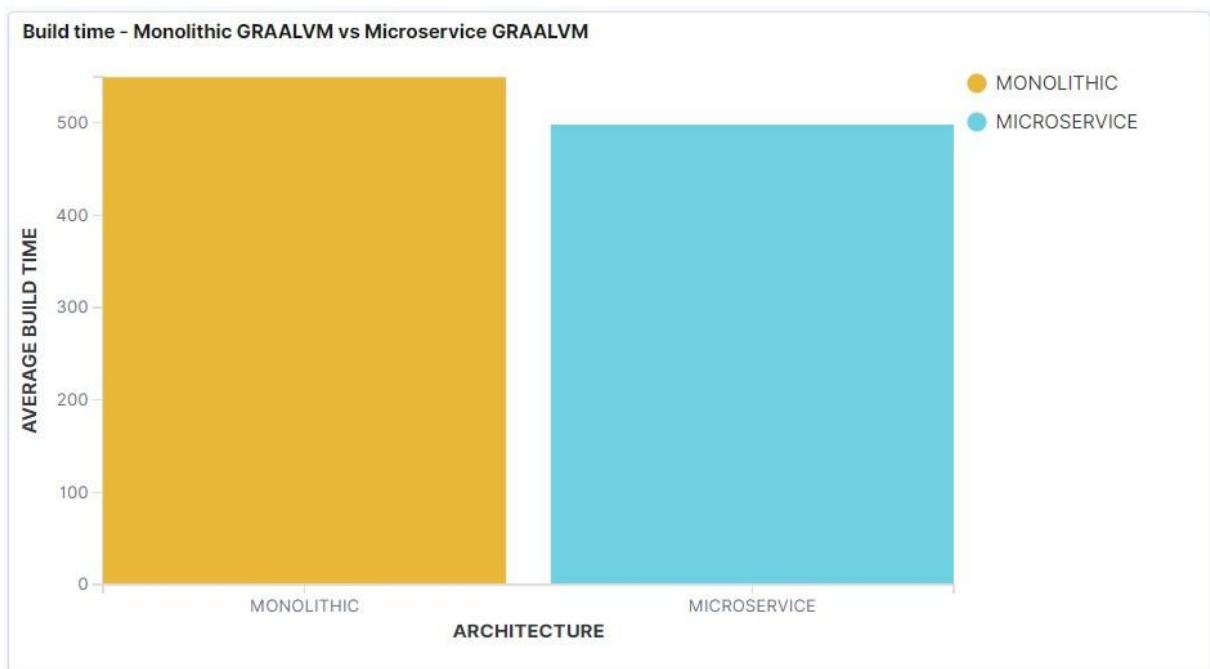


Слика 100. Проценат успешности - микросервисна (GraalVM) апликација

Са друге стране, микросервисна апликације је при оптерећењу од 100 и 1000 захтева забележила исти резултат, док је за 10000 захтева забележена успешност од 60%.

5.4.2.4. Време изградње извршног програма

На слици 101 је приказано време изградње извршног програма монолитне и микросервисне апликације на GraalVM виртуелној машини.



Слика 101. Време изградње извршног програма - монолитна (GraalVM) и микросервисна (GraalVM) апликација

За изградњу извршног програма обе апликације показују приближан резултат, са минималном предношћу микросервисне апликације којој је у просеку потребно 498.267 секунди у односу на 549.667 секунде које бележи монолит.

5.4.2.5. Време покретања апликације

На слици 102 је приказано време покретања монолитне и микросервисне апликације на GraalVM виртуелној машини.

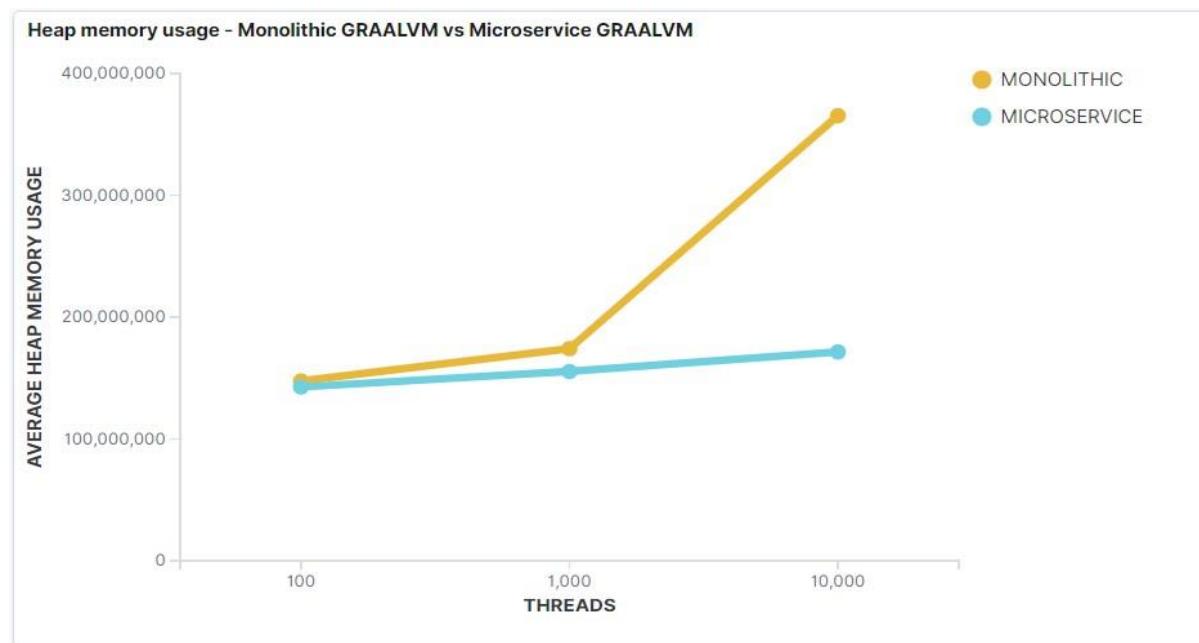


Слика 102. Време покретања апликације - монолитна (GraalVM) и микросервисна (GraalVM) апликација

Обе апликације показују приближан резултат просечног времена потребног за њихово покретање, при чему благу предност имају микросервиси. За покретање микросервисне апликације је у просеку потребно 0.233 секунди у односу на 0.28 секунде које бележи монолит.

5.4.2.6. Употреба heap меморије

На слици 103 је приказана употреба heap меморије монолитне и микросервисне апликације на GraalVM виртуелној машини.

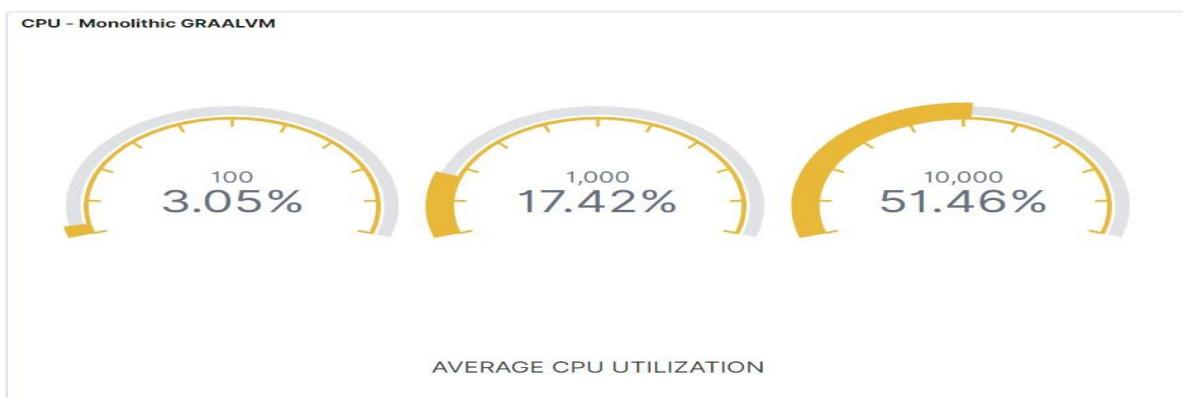


Слика 103. Употреба heap меморије - монолитна (GraalVM) и микросервисна (GraalVM) апликација

Са растом оптерећења, микросервисна апликација бележи благ, константан раст употребе heap меморије. Монолит, за оптерећење од 100 и 1000 захтева, показује приближне, незнатно веће резултате, док при оптерећењу од 10000 захтева бележи нагли пораст употребе heap меморије која је приближно 2 пута већа у односу на микросрвисе. Конкретно, за наведена оптерећења респективно, микросрвиси бележе резултате од 142,637,793.28, 155,346,272.296 и 171,410,095.537 байтова, у односу на 147,598,904.96, 174,093,454.489 и 365,424,346.76 байтова монолитне апликације.

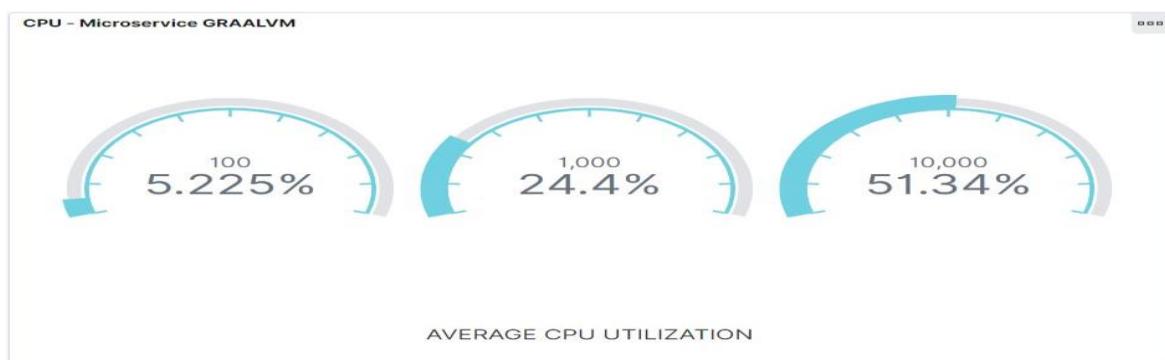
5.4.2.7. Употреба процесора

На слици 104 је приказана употреба ресурса процесора монолитне апликације на GraalVM виртуелној машини.



Слика 104. Употреба процесора - монолитна (GraalVM) апликација

Монолитна апликација, за оптерећења од 100, 1000 и 10000 захтева, бележи употребу процесора од 3.05%, 17.42% и 51.46% респективно. На слици 105 је приказана употреба ресурса процесора микросрвисне апликације на GraalVM виртуелној машини.



Слика 105. Употреба процесора - микросрвисна (GraalVM) апликација

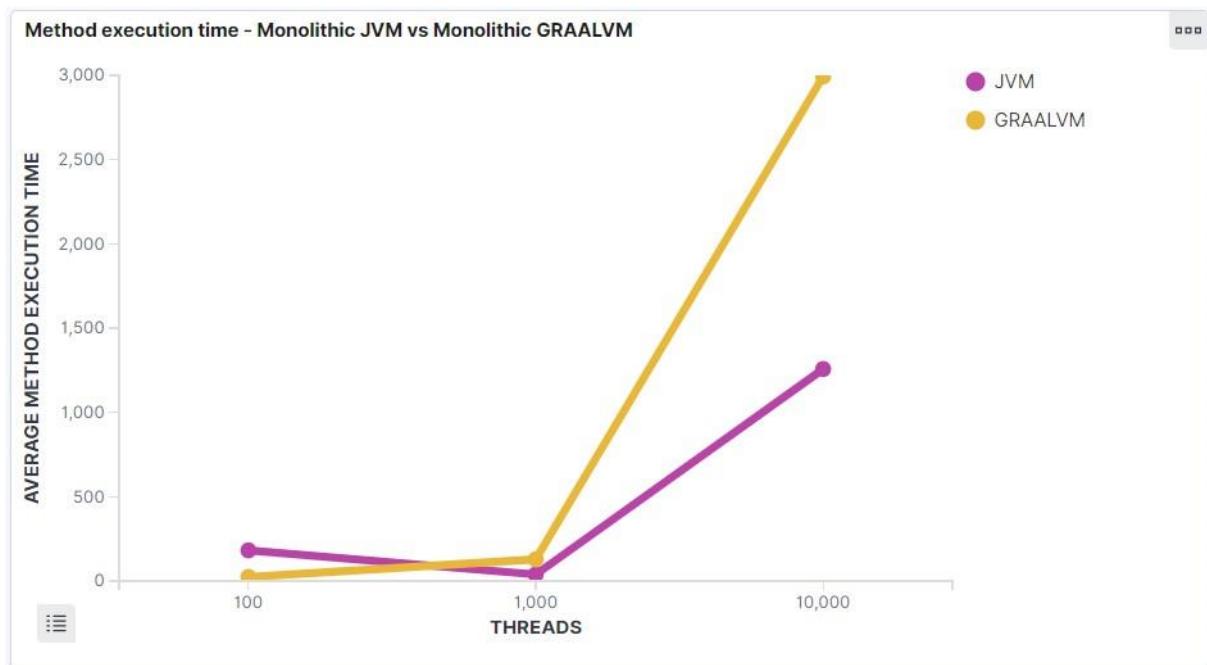
Микросрвисна апликација захтева приближну количину ресурса процесора, за све наведене количине захтева, са забележеним резултатима од 5.225%, 24.4% и 51.34% респективно. Блага предност монолитне апликације од 6.98% се примећује при средњој количини оптерећења.

5.4.3. Поређење монолитне (JVM) и монолитне (GraalVM) апликације

У оквиру овог поглавља је извршена упоредна анализа добијених резултата монолитне (JVM) и монолитне (GraalVM) апликације.

5.4.3.1. Време извршења кључних метода

На слици 106 је приказано време извршења кључних метода монолитних апликација на Јава и GraalVM виртуелној машини.

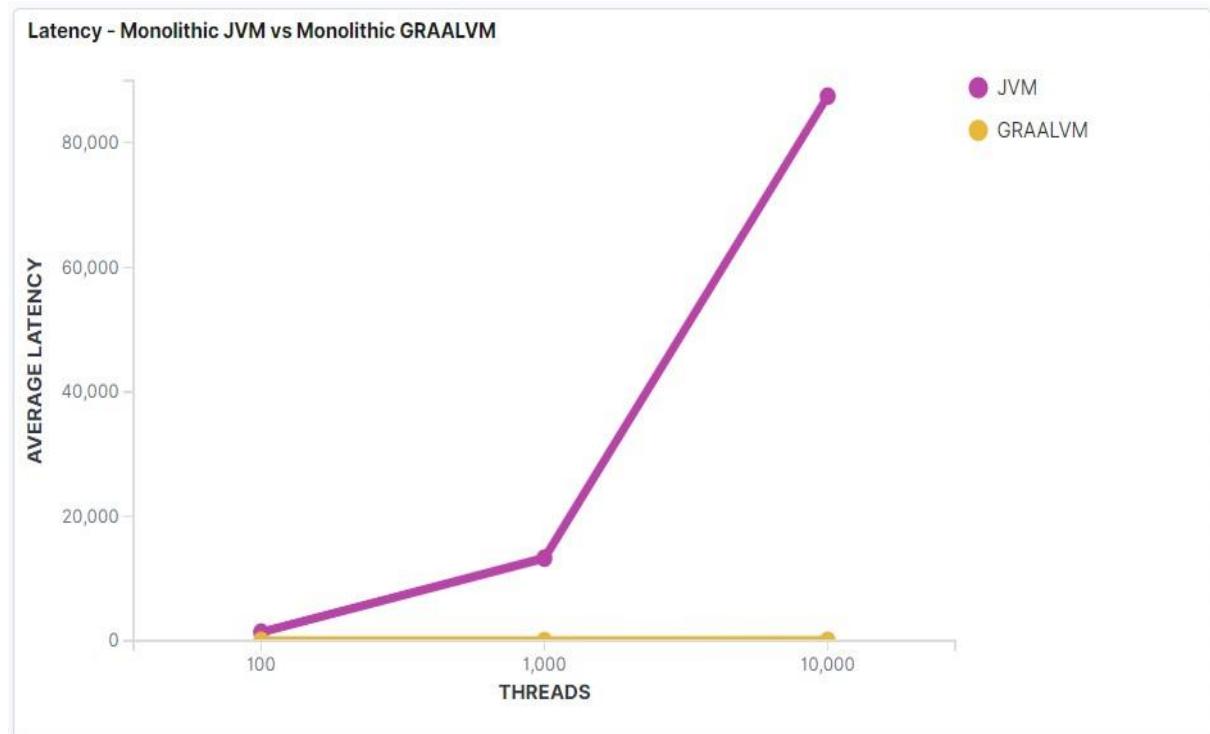


Слика 106. Време извршења - монолитна (JVM) и монолитна (GraalVM) апликација

При оптерећењу од 100, 1000 и 10000 захтева, монолитна апликација која користи JVM бележи просечно време извршења метода од 181.76, 39.264 и 1,257.508 милисекунди респективно, док апликација која користи GraalVM постиже резултат од 24.27, 128.549 и 2,992.59 милисекунди. Може се закључити да GraalVM верзија има предност при малој количини оптерећења, док са порастом броја захтева боље резултате пружа JVM.

5.4.3.2. Кашњење

На слици 107 је приказано измерено кашњење монолитних апликација на Јава и GraalVM виртуелној машини.



Слика 107. Кашњење - монолитна (JVM) и монолитна (GraalVM) апликација

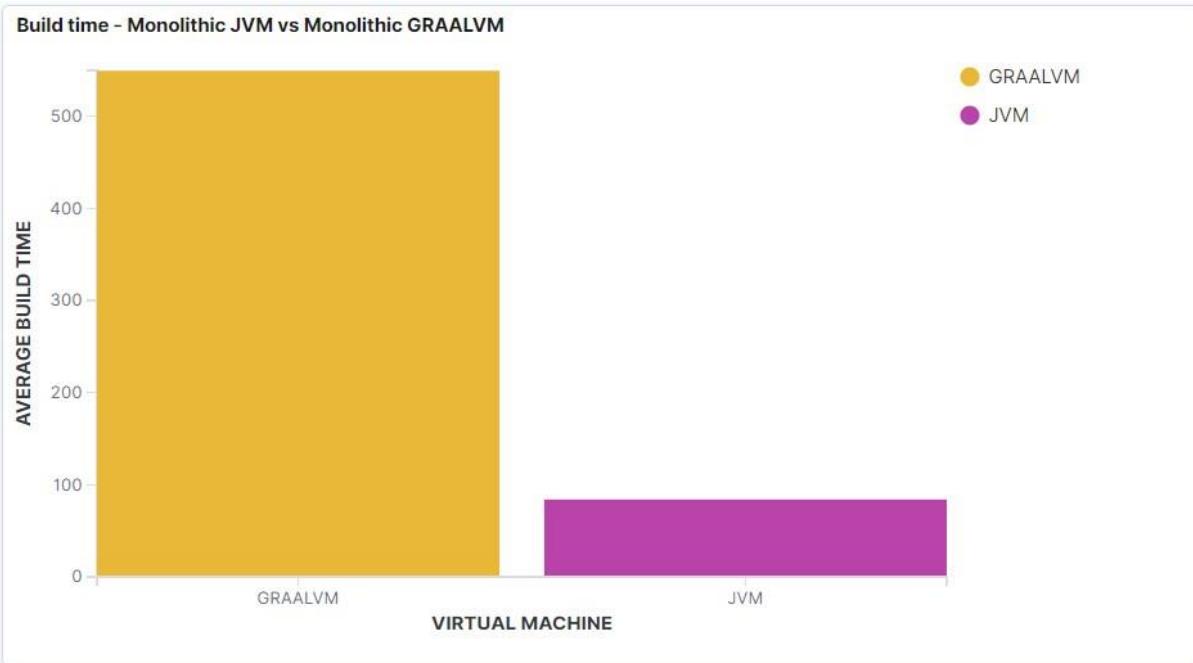
При оптерећењу од 100, 1000 и 10000 захтева, монолитна апликација која користи JVM бележи просечно кашњење од 1,381.74, 13,275.424 и 87,499.284 милисекунди респективно, док апликација која користи GraalVM постиже резултат од 56.31, 29.797 и 66.852 милисекунди. Може се закључити да GraalVM показује изразито боље резултате, чија је просечна вредност кашњења константна независно од броја корисничких захтева. Са друге стране, код JVM апликације се примећује директан утицај повећања броја корисничких захтева на раст времена просечног кашњења.

5.4.3.3. Проценат успешности

По вредностима приказаним на сликама 90 и 99, може се закључити да обе апликације показују исте или приближне резултате процента успешности, независно од количине оптерећења.

5.4.3.4. Време изградње извршног програма

На слици 108 је приказано време изградње извршног програма монолитних апликација на Јава и GraalVM виртуелној машини.

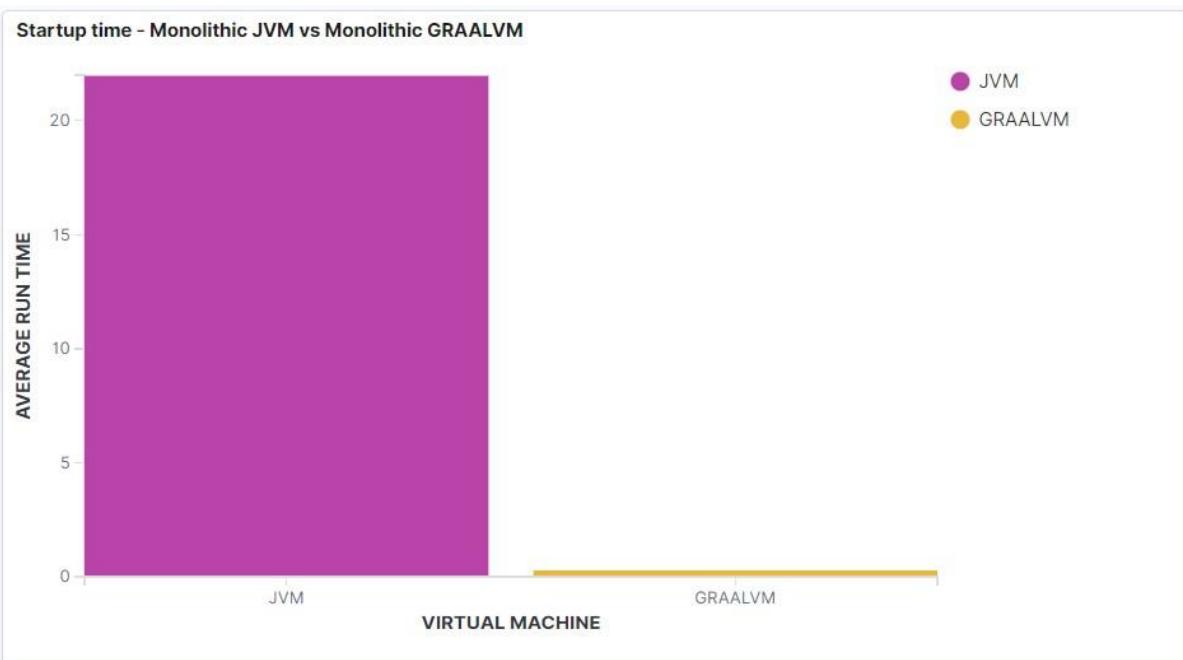


Слика 108. Време изградње извршног програма - монолитна (JVM) и монолитна (GraalVM) апликација

За изградњу извршног програма апликације која користи JVM у просеку је потребно 84 секунде, што је око 6.5 пута бољи резултат у односу на 549.667 секунди које су потребне за изградњу извршног програма апликације употребом GraalVM.

5.4.3.5. Време покретања апликације

На слици 109 је приказано време покретања монолитних апликација на Јава и GraalVM виртуелној машини.

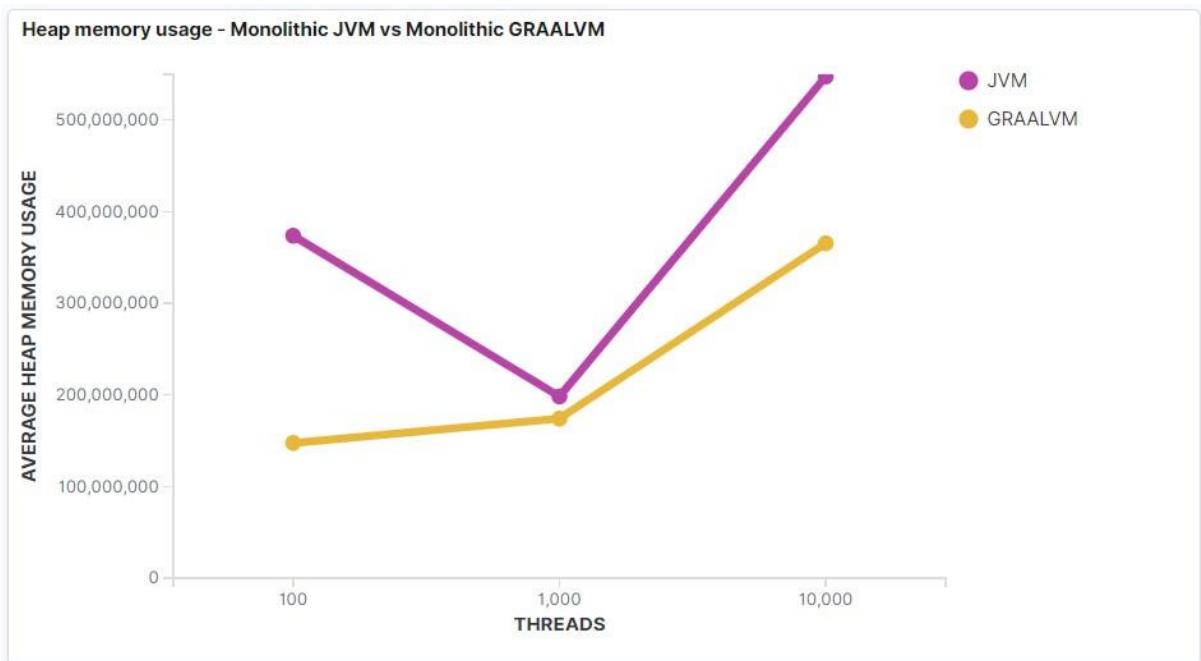


Слика 109. Време покретања апликације - монолитна (JVM) и монолитна (GraalVM) апликација

За покретање JVM апликације је потребно 21.967 секунди, док се апликација која користи GraalVM у просеку покреће брзином од 0.28 секунди, што је 78.4 пута бољи резултат.

5.4.3.6. Употреба heap меморије

На слици 110 је приказана употреба heap меморије монолитних апликација на Јава и GraalVM виртуелној машини.



Слика 110. Употреба heap меморије - монолитна (JVM) и монолитна (GraalVM) апликација

Апликација која користи GraalVM показује боље резултате, мањом употребом heap меморије, независно од количине оптерећења. Са порастом броја захтева расте и њена просечна употреба меморије, бележећи резултате од 147,598,904.96, 174,093,454.489 и 365,424,346.76 бајтова за 100, 1000 и 10000 захтева респективно. Са друге стране, монолитна апликација која користи JVM захтева просечну употребу од 373,775,873.413, 198,255,508.332 и 547,677,220.182. Може се приметити да овој апликацији највише одговара средња количина оптерећења.

5.4.3.7. Употреба процесора

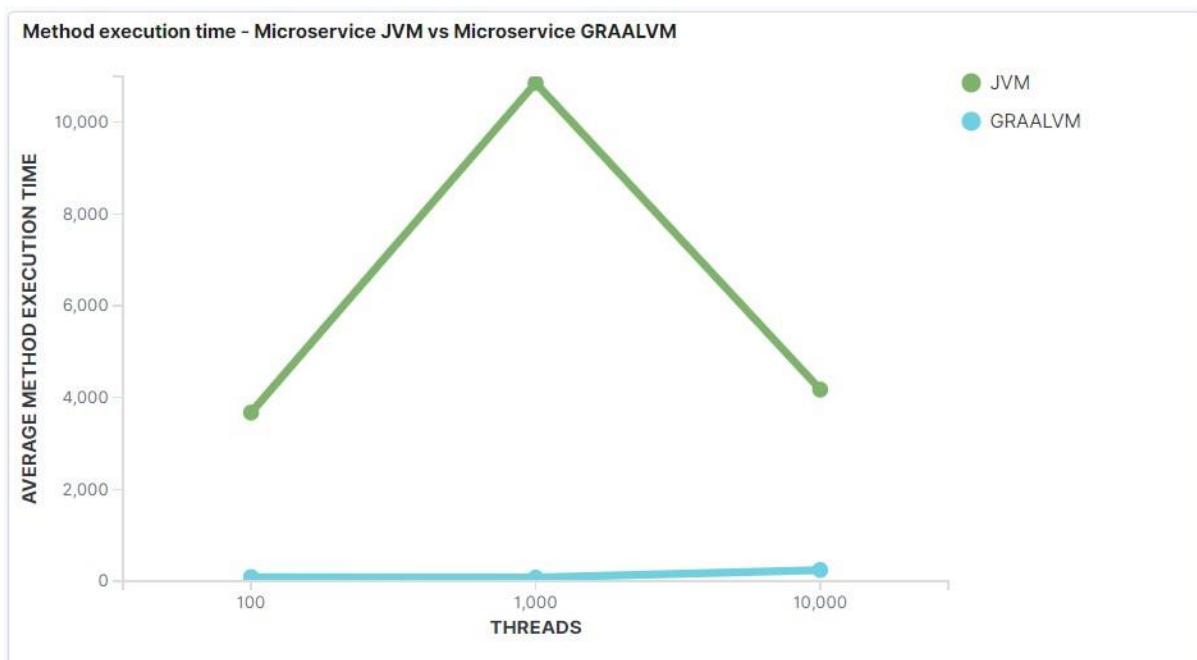
По вредностима приказаним на сликама 95 и 104 се може закључити да, независно од количине оптерећења, апликација која користи GraalVM у просеку захтева мању употребу процесора. При томе, највећа разлика се примећује при малом оптерећењу од 100 захтева, где GraalVM верзија апликације у просеку захтева 5 пута мању количину ресурса процесора у односу на апликацију која користи JVM.

5.4.4. Поређење микросервисне (JVM) и микросервисне (GraalVM) апликације

У оквиру овог поглавља је извршена упоредна анализа добијених резултата микросервисне (JVM) и микросервисне (GraalVM) апликације.

5.4.4.1. Време извршења кључних метода

На слици 111 је приказано време извршења кључних метода микросервисних апликација на Јава и GraalVM виртуелној машини.

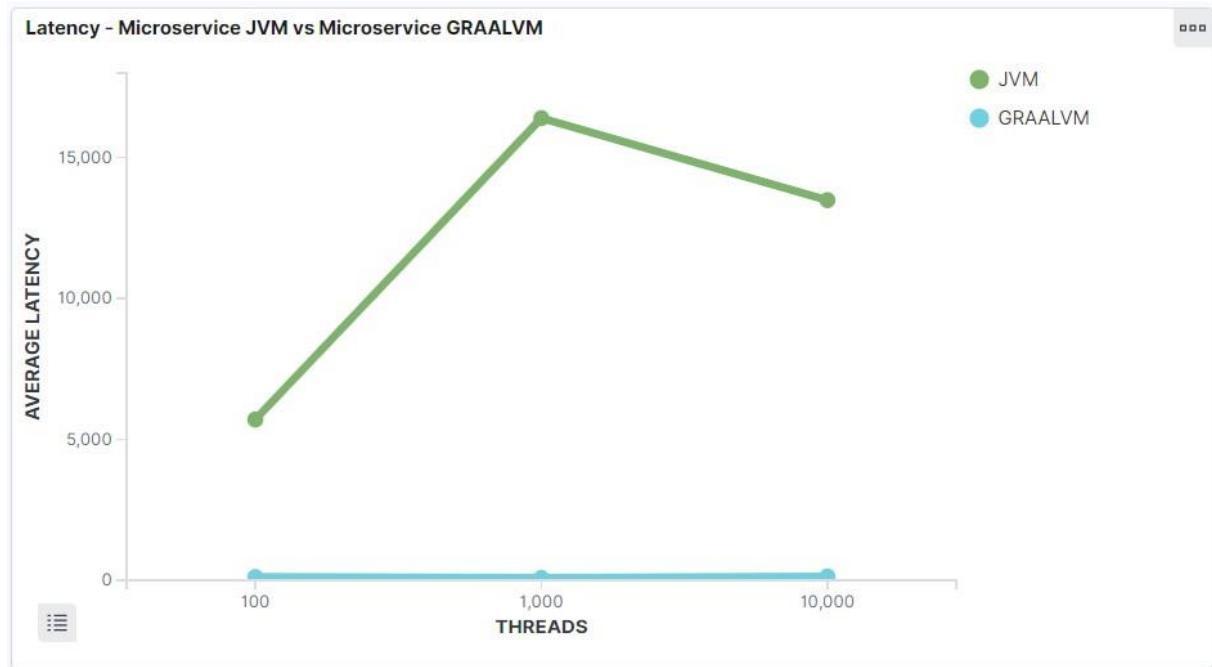


Слика 111. Време извршења - микросервисна (JVM) и микросервисна (GraalVM) апликација

Микросервисна апликација која користи GraalVM је показала вишеструко боље резултате, независно од оптерећења. Она при оптерећењу од 100, 1000 и 10000 корисничких захтева показује приближне резултате просечног времена извршења метода од 89.89, 79.59 и 243.673 милисекунди респективно, док је за исти број захтева апликацији која користи JVM потребно 3,673.629, 10,858.063 и 4,175.564 милисекунди. Може се приметити да JVM верзија апликације има приближна просечна времена извршења за 100 и 10000 захтева, док при оптерећењу од 1000 захтева бележи нагли раст.

5.4.4.2. Кашњење

На слици 112 је приказано измерено кашњење микросервисних апликација на Јава и GraalVM виртуелној машини.



Слика 112. Кашњење - микросервисна (JVM) и микросервисна (GraalVM) апликација

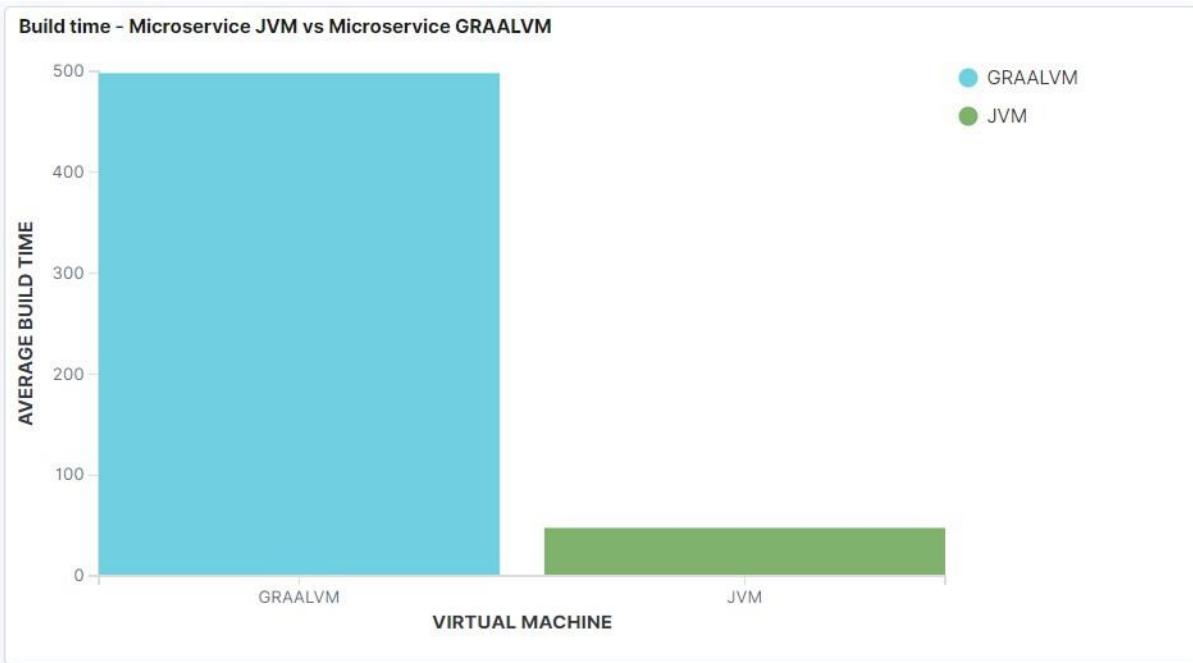
Микросервисна апликација која користи GraalVM бележи константна и изузетно мала просечна времена кашњења од 117.8, 82.115 и 134.303 милисекунди при оптерећењу од 100, 1000 и 10000 захтева респективно. Са друге стране, апликација која користи JVM показује знатно лошије резултате од 5,705.89, 16,401.315 и 13,490.625 милисекунди за исту количину захтева. При оптерећењу од 1000 захтева се примећује највећа разлика, где GraalVM верзија апликације постиже приближно 200 пута боље резултате просечног кашњења у поређењу са JVM верзијом.

5.4.4.3. Проценат успешности

По вредностима приказаним на сликама 91 и 100, може се закључити да обе апликације бележе стопостотан проценат успешности при оптерећењу од 100 и 1000 захтева. При оптерећењу од 10000 корисничких захтева, апликација која користи JVM показује изузетно лош резултат од 37.39%, док GraalVM верзија апликације бележи бољи али и даље не доволјно добар резултат од 60%.

5.4.4.4. Време изградње извршног програма

На слици 113 је приказано време изградње извршног програма микросервисних апликација на Јава и GraalVM виртуелној машини.

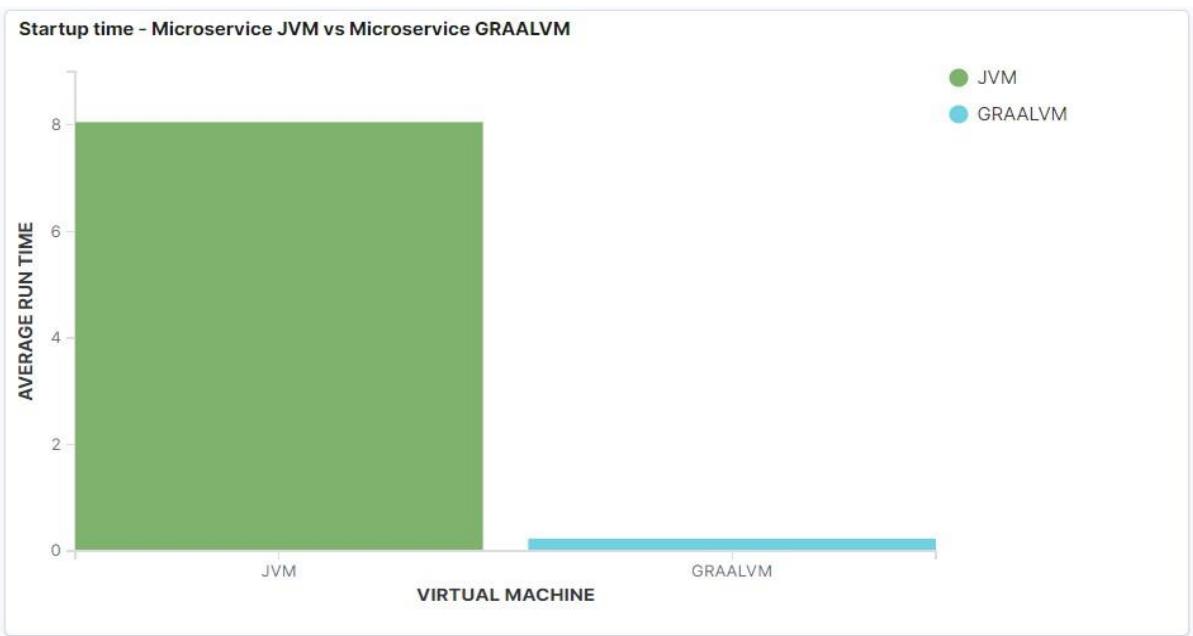


Слика 113. Време изградње извршног програма - микросервисна (JVM) и микросервисна (GraalVM) апликација

За изградњу извршног програма апликације која користи JVM у просеку је потребно 47.641 секунди, што је око 10 пута бољи резултат у односу на 498.267 секунди које су потребне за изградњу извршног програма апликације употребом GraalVM.

5.4.4.5. Време покретања апликације

На слици 114 је приказано време покретања микросервисних апликација на Јава и GraalVM виртуелној машини.

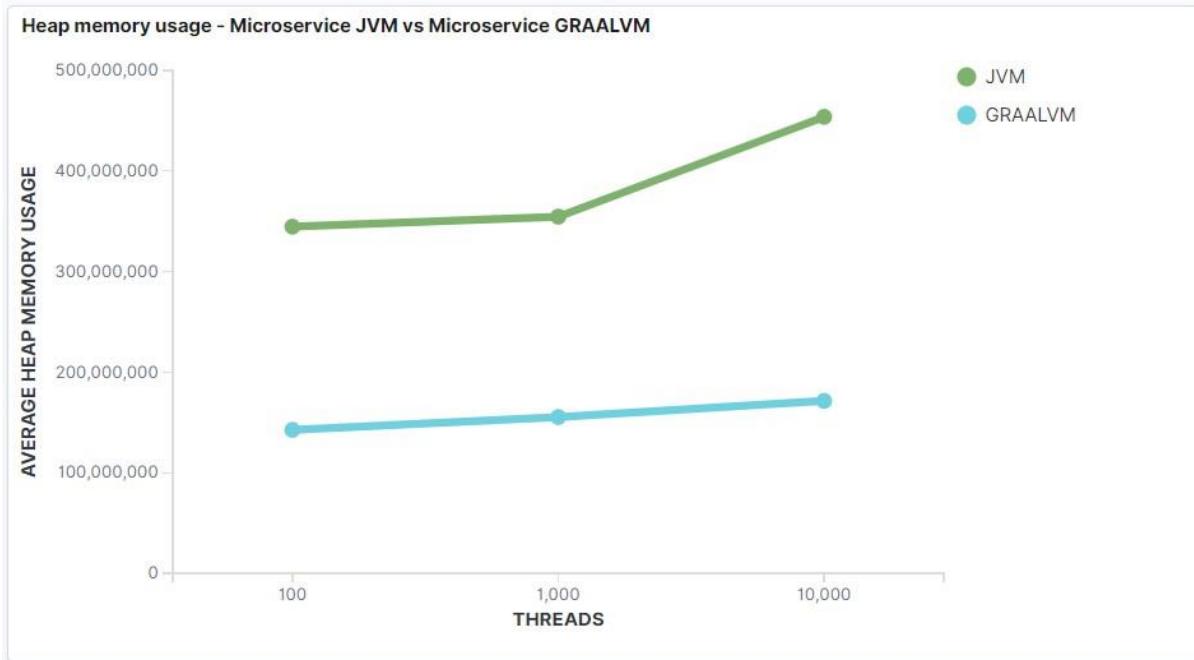


Слика 114. Време покретања апликације - микросервисна (JVM) и микросервисна (GraalVM) апликација

За покретање JVM апликације је потребно 8.053 секудни, док се апликација која користи GraalVM у просеку покреће брзином од 0.233 секунди, што је 34.5 пута бољи резултат.

5.4.4.6. Употреба heap меморије

На слици 115 је приказана употреба heap меморије микросрвисних апликација на Јава и GraalVM виртуелној машини.



Слика 115. Употреба heap меморије - микросрвисна (JVM) и микросрвисна (GraalVM) апликација

Апликација која користи GraalVM показује боље резултате, приближно 2.4. пута мањом просечном употребом heap меморије, независно од количине оптерећења. Са порастом броја захтева обе апликације бележе раст просечне употребе меморије. GraalVM верзија апликације у просеку користи 142,637,793.28, 155,346,272.296 и 171,410,095.537 байтова за 100, 1000 и 10000 захтева респективно. За исти број захтева, апликација која користи JVM бележи просечну употребу меморије од 344,660,001.412, 354,620,017.92 и 453,854,881.578 байтова.

5.4.4.7. Употреба процесора

По вредностима приказаним на сликама 96 и 105 се може закључити да, независно од количине оптерећења, апликација која користи GraalVM у просеку захтева мању употребу процесора. При томе, највећа разлика се примећује при малом оптерећењу од 100 захтева, где GraalVM верзија апликације у просеку захтева 6.5 пута мању количину ресурса процесора у односу на апликацију која користи JVM.

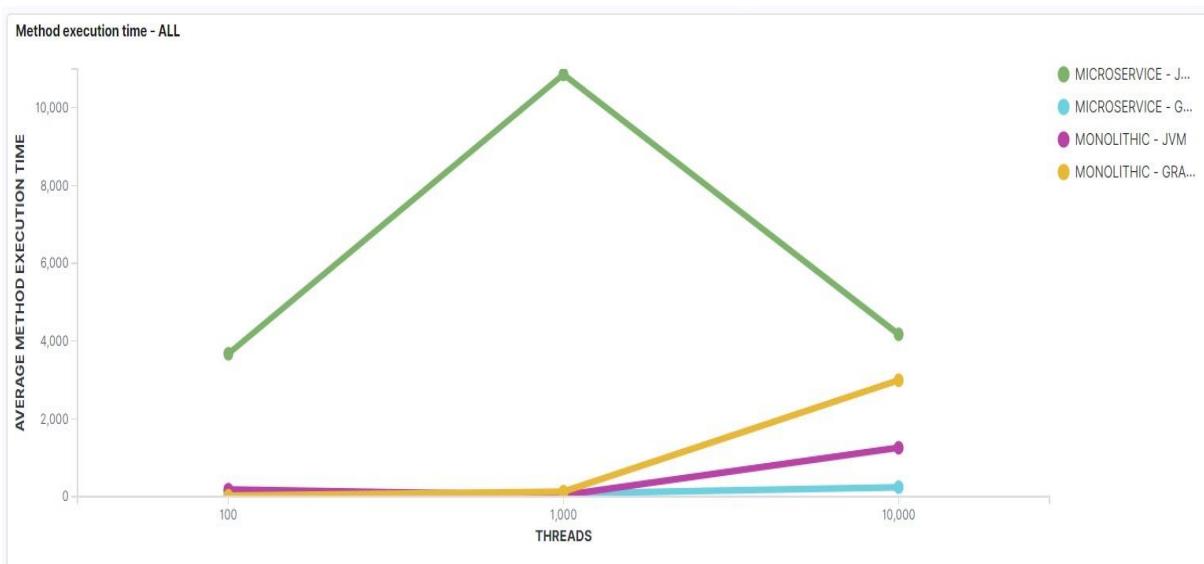
5.5. Дискусија

У оквиру овог поглавља је дат закључак анализе упоредним прегледом добијених резултата свих верзија апликације на једном месту:

- Монолитна (JVM)
- Микросервисна (JVM)
- Монолитна (GraalVM)
- Микросервисна (GraalVM)

5.5.1. Време извршења кључних метода

На слици 116 је приказано време извршења кључних метода монолитних и микросервисних апликација на Јава и GraalVM виртуелној машини.

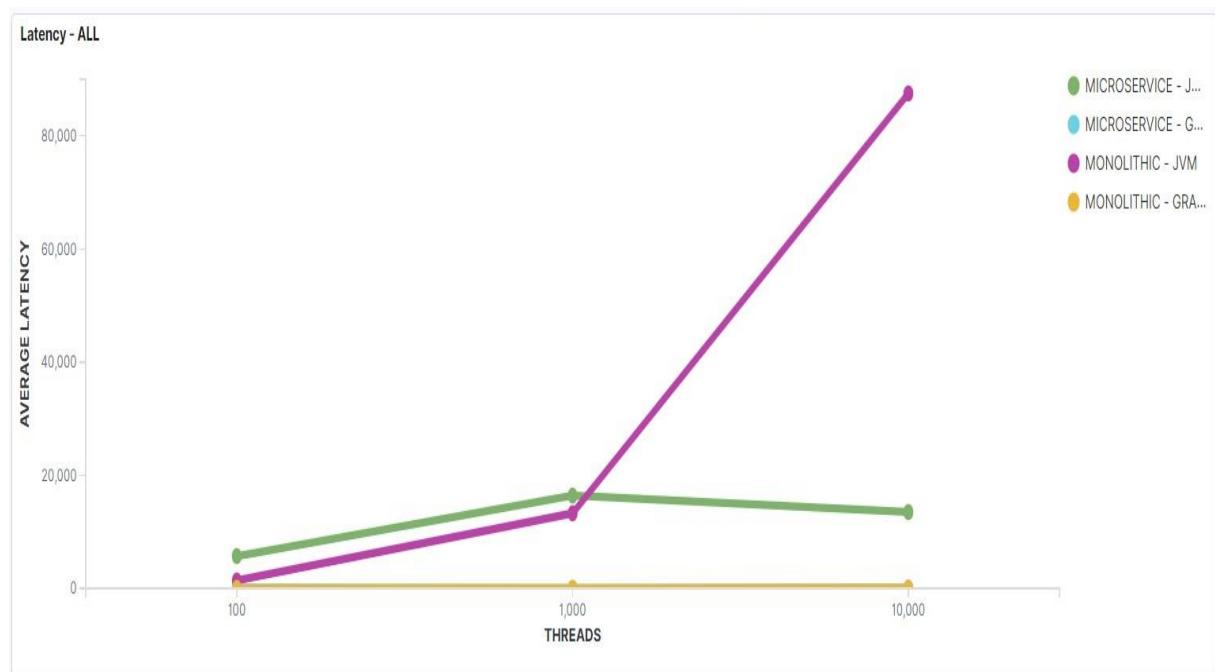


Слика 116. Време извршења – све апликације

Независно од броја корисничких захтева, најбоље резултате времена извршења кључних метода је постигла микросервисна апликација која користи GraalVM, док убедљиво најлошије резултате бележи микросервисна апликација применом JVM. При оптерећењу од 100 и 1000 захтева монолитна (JVM), монолитна (GraalVM) и микросервисна (GraalVM) апликација имају приближне резултате што не важи и за највеће тестирано оптерећење од 10000 захтева где значајно расте просечно време извршења кључних метода које бележе монолитне апликације обе виртуелне машине. На основу свега наведеног се може извући закључак да, уколико је време извршења метода од пресудног значаја за рад система, треба избегавати микросервисне (JVM) апликације. Уколико је очекивано оптерећење система мањег или средњег интензитета, не постоји значајна разлика између преостале три апликације, док се за велике очекиване количине оптерећења препоручује микросервисна (GraalVM) апликација.

5.5.2. Кашњење

На слици 117 је приказано измерено кашњење монолитних и микросервисних апликација на Java и GraalVM виртуелној машини.



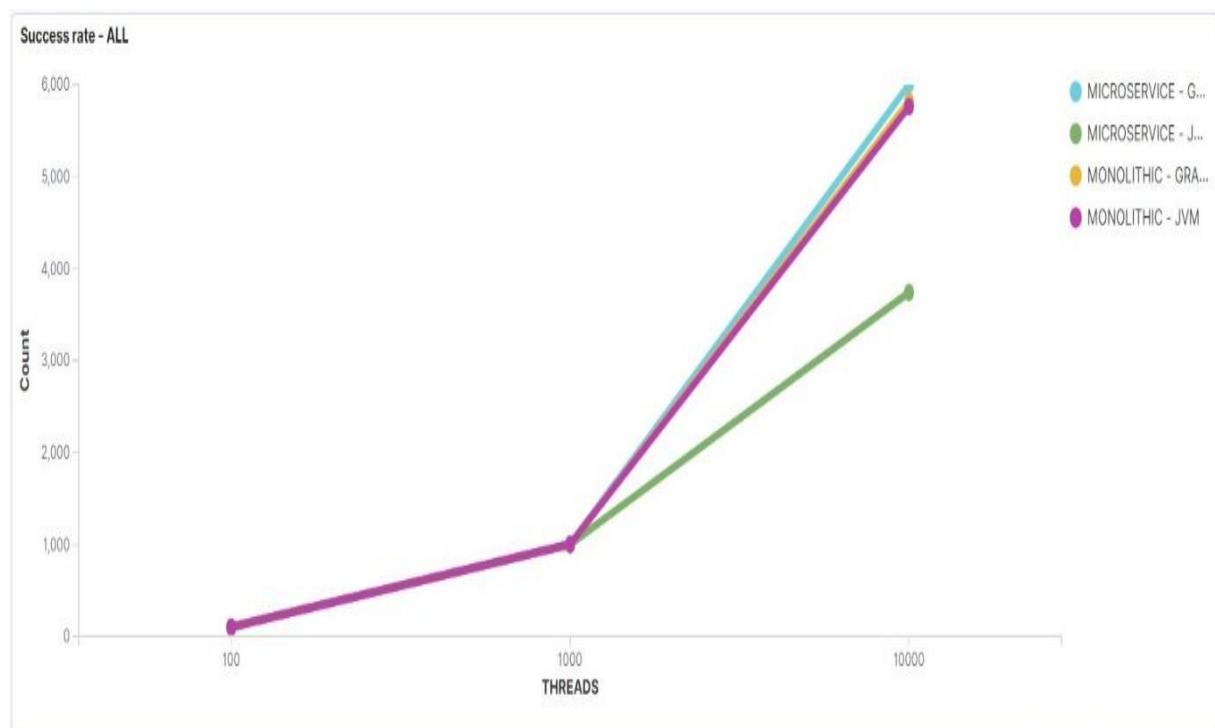
Слика 117. Кашњење - све апликације

Уколико је циљ постићи што мање време кашњења у раду система, најбоље је изабрати апликације које користе GraalVM, независно од архитектуре. Уколико се ипак бира између апликација које користе JVM, при малом и средњем оптерећењу између монолита и микросрвиса треба изабрати монолит, док се при великим оптерећењима препоручује микросрвисна архитектура.

Истраживање “Microservices vs. Monoliths: An Operational Comparison”, које не обухвата виртуелне машине, такође долази до закључка да монолитне апликације бележе мање време кашњења у поређењу са микросрвисним, независно од броја захтева. [46]

5.5.3. Проценат успешности

На слици 118 је приказан измерени проценат успешности монолитних и микросервисних апликација на Java и GraalVM виртуелној машини.



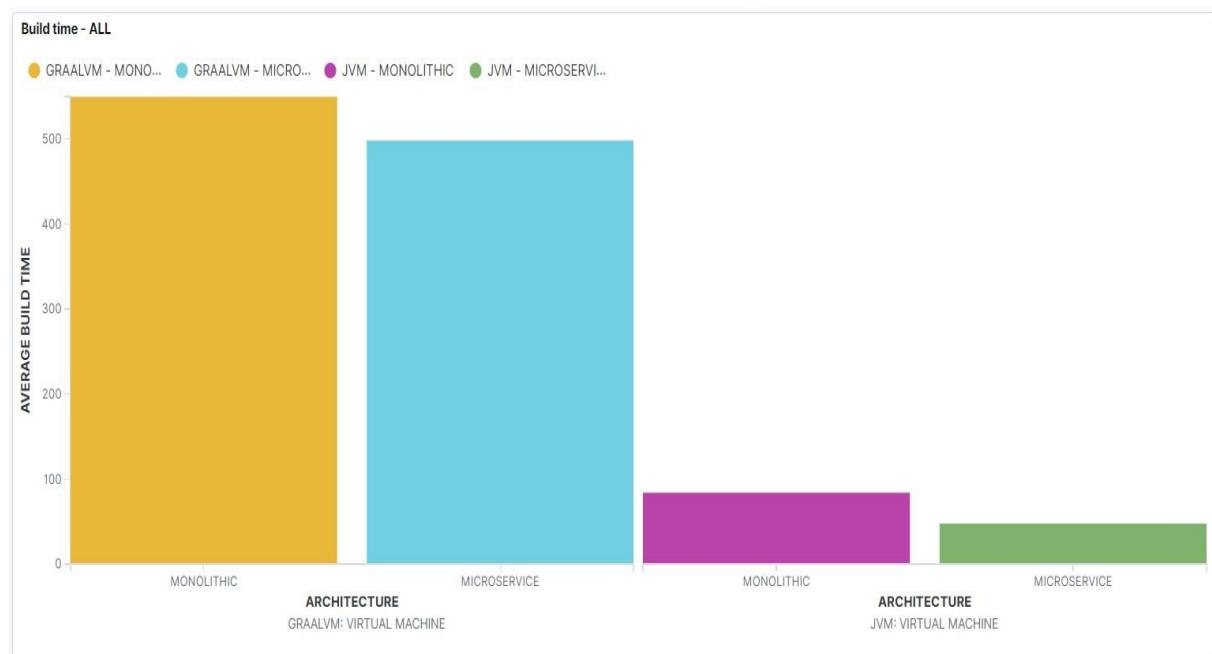
Слика 118. Проценат успешности - све апликације

Проценат успешности свих тестиралих апликација је приближен независно од количине оптерећења, осим за микросервисну (JVM) апликацију коју треба избегавати уколико се очекују оптерећења великог интензитета.

У оквиру рада “Performance characteristics between monolithic and microservice-based systems”, који не обухвата виртуелне машине, се такође бележе приближни проценти успешности монолита и микросрвиса при оптерећењима од 1, 10 и 100 захтева, док при порасту оптерећења на 1000 захтева микросрвиси такође показују знатно лошије резултате.[47]

5.5.4. Време изградње извршног програма

На слици 119 је приказано време изградње извршног програма монолитних и микросервисних апликација на Јава и GraalVM виртуелној машини.

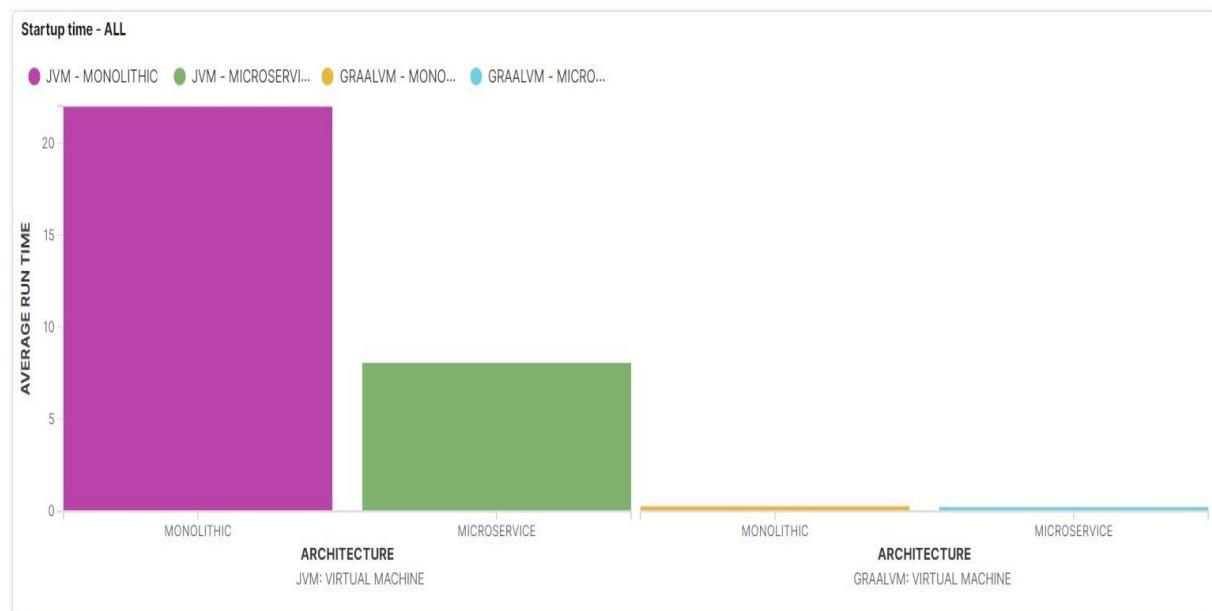


Слика 119. Време изградње извршног програма – све апликације

Време изградње извршног програма је критеријум на који у овом случају највише утиче избор виртуелне машине. Са тим у вези, боље резултате су постигле апликације које користе JVM, што је и очекивано због њеног JIT (Just In Time) компајлера. Узрок спорије изградње извршних програма апликација које користе GraalVM јесте AOT (Ahead Of Time) компајлирање на коме се базира GraalVM Spring Native технологија и које знатно дуже траје јер се унапред компајлирају све класе и библиотеке које су неопходне за моментално извршавање апликације.

5.5.5. Време покретања апликације

На слици 120 је приказано време покретања монолитних и микросервисних апликација на Јава и GraalVM виртуелној машини.



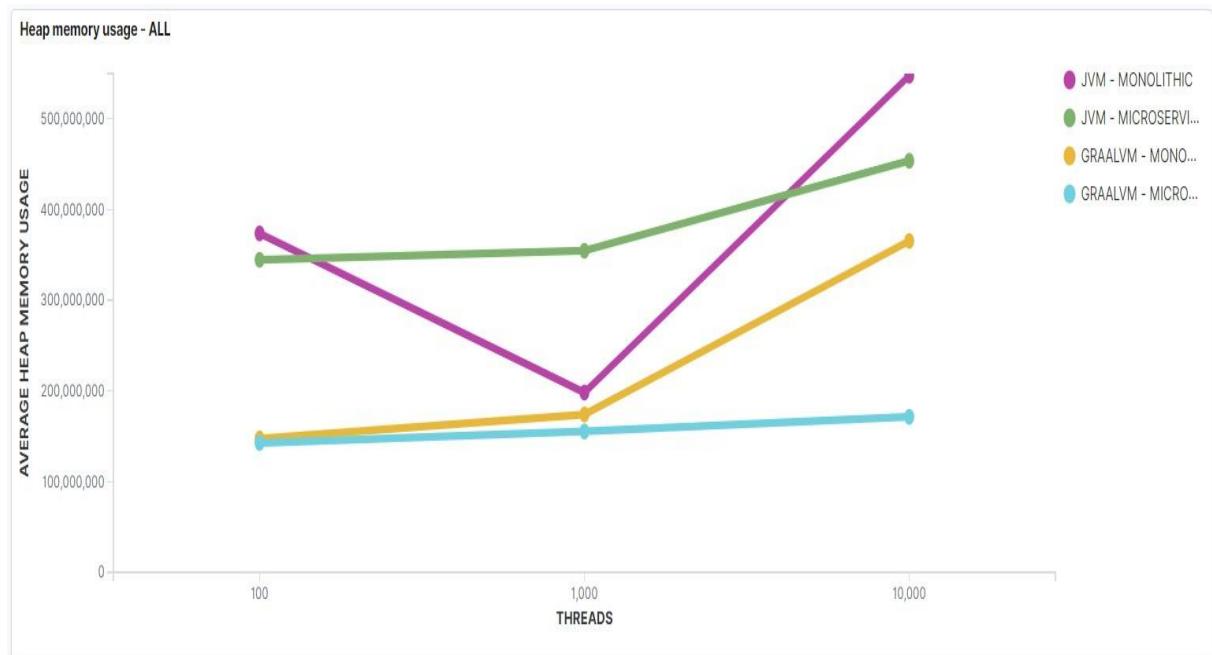
Слика 120. Време покретања апликације – све апликације

Време покретања апликације је такође критеријум на који највише утиче избор виртуелне машине. За софтверске системе којима је неопходно што брже покретање је најбоље решење примена Graal виртуелне машине, независно од архитектуре. Разлог за то је чињеница да је њен АОТ компајлер већ извршио процес превођења у машински код пре самог покретања апликације.

Бољи резултат времена покретања апликација на GraalVM виртуелној машини у поређењу са Јава виртуелном машином бележи и истраживање “Benchmarking Web Services using GraalVM Native Image”, које не обухвата архитектуру апликација. [48]

5.5.6. Употреба heap меморије

На слици 121 је приказана употреба heap меморије монолитних и микросервисних апликација на Јава и GraalVM виртуелној машини.



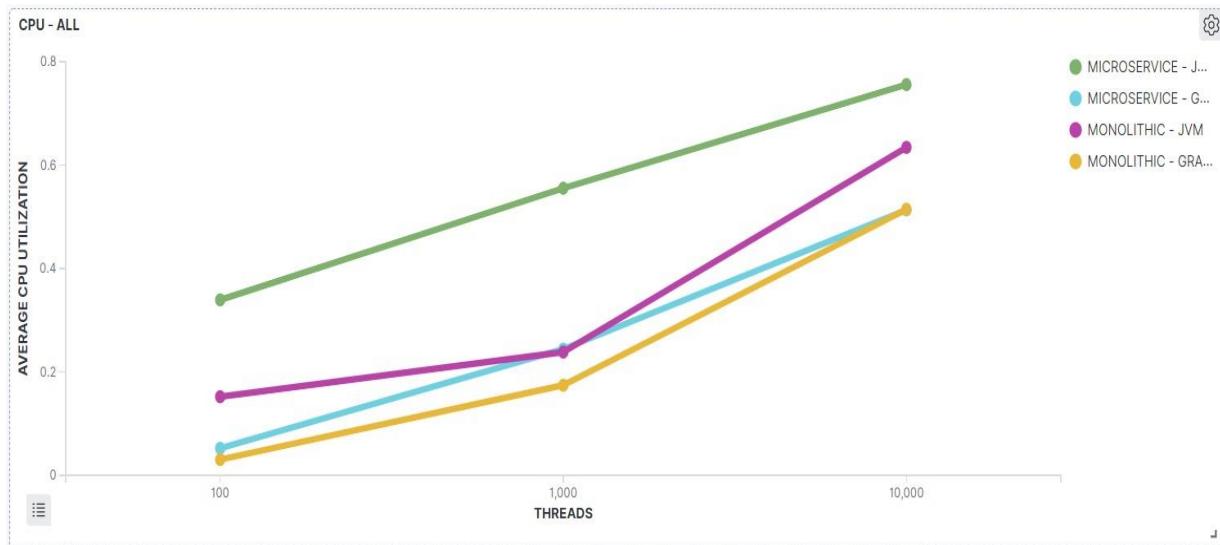
Слика 121. Употреба heap меморије – све апликације

Најбоље резултате у виду мале употребе heap меморије су оствариле апликације које користе Graal виртуелну машину, независно од архитектуре и броја захтева. Са тим у вези ова виртуелна машина представља прави избор у ситуацијама када се располаже ограниченим ресурсима. Због компајлирања које она обавља пре времена извршења програма, у току самог рада нема потребе за JIT компајлирањем што значајно ослобађа ресурсе меморије. Најбољи ефекат се постиже применом микросервисне архитектуре и GraalVM, који заједно бележе најмању употребу heap меморије од свих тестиралих опција. Уколико се бира између апликација које користе JVM, при малом и великом оптерећењу између монолита и микросервиса треба изабрати микросервисну архитектуру, док при средњој количини захтева монолитна архитектура показује знатно боље резултате.

Сличан закључак у поређењу употребе ресурса меморије апликација на GraalVM и Јава виртуелној машини бележи и истраживање “Benchmarking Web Services using GraalVM Native Image”, које не обухвата архитектуру апликација, где GraalVM показује знатно боље резултате. [48]

5.5.7. Употреба процесора

На слици 122 је приказана употреба ресурса процесора монолитних и микросервисних апликација на Јава и GraalVM виртуелној машини.



Слика 122. Употреба процесора – све апликације

Најбоље резултате по питању употребе ресурса процесора су показале апликације које користе Graal виртуелну машину из истих разлога који су наведени за употребу heap меморије. У раду са овом виртуелном машином, монолитна и микросервисна апликација су постигле приближне резултате при малом и великом оптерећењу, док се под средњом количином захтева боље показала монолитна архитектура. Између апликација које користе JVM се боље показала монолитна, док микросервисну треба избегавати уколико је систем покренут у окружењу са ограниченој количином ресурса. По измереним резултатима се може закључити да повећање корисничких захтева директно утиче повећање употребе ресурса процесора, независно од архитектуре и виртуелне машине која се користи.

5.6. Ограничења евалуације

Важно је напоменути да су на евалуацију која је извршена у оквиру овог рада утицала следећа ограничења:

- Разматран је само један софтверски систем, при чему нису узети у обзир системи другачијег нивоа сложености или домена.
- У зависности од проблема, одлуке о декомпозицији монолитне апликације могу бити другачије. Није узет у обзир утицај осталих могућих стратегија декомпозиције и дефинисања граница између микросервиса.
- Истраживање није употребљено додатним метрикама јер коришћени алати не подржавају њихово мерење у свим развијеним верзијама апликације. Једна од њих је свакако време прикупљања смећа (*garbage collection time*) која је директно везана за рад виртуелних машина. Она није укључена у истраживање зато што помоћу коришћених технологија није било могуће мерење вредности ове метрике за апликације које користе GraalVM, у тренутку писања.
- Spring Native технологија се, у тренутку писања, налази у експерименталној фази и није компатибилна са одређеним библиотекама. Из тог разлога у верзијама апликације које користе GraalVM нису примењени Spring Security и Liquibase док у апликацијама које користе Јава виртуелну машину јесу.
- Због хардверских ограничења нису вршена тестирања под већом количином оптерећења која би употребнила истраживање.
- Примењени JMeter тест планови нису прилагођени појединачним метрикама.
- Нису узети у обзир остали JVM језици као што су Kotlin, Groovy и остали.
- Радни оквири коришћени за развој монолита и микросервиса могу бити другачији. Поред радног оквира Spring Boot, добар избор би били и Quarkus и Micronaut, који су интегрисани са GraalVM. [11]
- Механизми комуникације микросервиса могу бити другачији. На одређеним местима је могућа асинхронна комуникација применом технологија као што су ActiveMQ, Kafka и друге.

6. Закључак

У оквиру овог рада је представљен упоредни преглед карактеристика монолитних и микросервисних апликација у Јава ЕЕ окружењу. Апликације клијентске и серверске стране које су коришћене за потребе тестирања су имплементиране неким од најсавременијих технологија, као што су Spring Boot и његови модули, Angular и остale, које су детаљно описане у уводном делу рада. За бележење и мерење резултата тестирања су коришћене технологије JMeter и Elastic Stack. Предмет истраживања у великој мери обухвата и рад монолитних и микросервисних система са Јава и GraalVM виртуелним машинама. Са тим у вези су објашњени начини њиховог функционисања, компоненте од којих су сачињене као и механизми компајлирања пре и у време извршења програма, заједно са бенефитима које они пружају. Посебна пажња је посвећена још увек експерименталној Spring Native технологији која пружа подршку за компајлирање Spring апликација помоћу GraalVM native image компајлера.

Темељно су теоријски описана своства монолитне и микросервисне архитектуре, заједно са њиховим предностима и недостацима. Затим је комплетно објашњен поступак миграције монолитне у микросервисну архитектуру. Представљени су разлози због којих се све већи број организација одулучује на овај корак, који се све циљеви тиме могу постићи и које су кључне стратегије миграције које се користе у пракси. Такође је објашњен и начин на који овај процес утиче на саму организациону структуру и појединце који је спроводе. Посебно је истакнут значај инкременталног приступа у декомпоновању монолитних софтверских система. Поред декомпозиције апликативног кода, велика пажња је посвећена и декомпоновању монолитне базе података које се сматра неопходним како би процес миграције био потпун. Описани су и изазови до којих често долази приликом спровођења овог процеса, заједно са практичним предлогима њиховог превазилажења.

Развој софтверског система, на студијском примеру Football Fantasy игре, је објашњен применом упрошћене Ларманове методе развоја софтвера. Детаљно је описана свака од њених пет фаза: прикупљање захтева од корисника, анализа, пројектовање, имплементација и тестирање. У фази прикупљања захтева од корисника, најпре је дат вербални опис модела, а затим је дефинисан модел случајева коришћења. Након тога је у фази анализе дат опис логичке структуре и понашања софтверског система, помоћу системских дијаграма секвенци, уговора о системским операцијама, концептуалног и релационог модела. У фази пројектовања је описана архитектура софтверског система тј. извршено је пројектовање корисничког интерфејса, апликационе логике и складишта података. Након тога је описана фаза имплементације. У последњој фази ове методе је описан начин на који је извршено тестирање апликације.

Централни део рада представља упоредна анализа употребе монолитне и микросервисне архитектуре у развоју софтверских система на практичном примеру. На самом почетку је дата формулатија проблема са предлогом решења. Описане су архитектуре и разлике између свих апликација које су развијене за потребе тестирања:

- Монолитна (JVM) апликација
- Микросервисна (JVM) апликација
- Монолитна (GraalVM) апликација
- Микросервисна (GraalVM) апликација

Након тога је прецизно описан поступак мерења резултата. Представљена су подешавања JMeter тест планова који су коришћени за симулацију интеракције корисника са системом. Детаљно су описани начини мерења и бележења резултата путем сопствене библиотеке развијене за потребе мониторинга, JMeter и Elastic Stack технологија. Метрике чије су вредности анализиране у оквиру тестирања су:

1. Време извршења кључних метода
2. Кашњење
3. Проценат успешности
4. Време изградње извршног програма
5. Време покретања апликације
6. Употреба heap меморије
7. Употреба процесора

Затим су текстуално и визуелно представљени добијени резултати међусобним поређењем претходно наведених, појединачних апликација по дефинисаном скупу критеријума. У закључку анализе су на једном месту упоређене све апликације по наведеним критеријумима, изнета су финална запажања, закључци и препоруке до којих се дошло анализом добијених резултата. Апликације које користе GraalVM су показале изузетно добрe резултате, поготово када је реч о времену покретања, кашњењу и системским метрикама као што су употреба ресурса меморије и процесора, док су са друге стране бележиле изузетно лоше време изградње извршних програма. Због експерименталне природе пројекта Spring Native, његове некомпатибилности са великим бројем постојећих библиотека и саме тежине имплементације, можда још увек није прави тренутак за његову употребу док не постане у потпуности стабилан. Апликације развијене применом микросервисне архитектуре су показале лоше перформансе када је реч о времену извршења метода, кашњењу и проценту успешности, што говори да је њихова популарност више заснована на флексибилности, слабој повезаности, могућностима независног скалирања и вршења измена.

Спроведено истраживање је могуће унапредити на разне начине. Пре свега, укључивањем додатних метрика у анализу би се добила потпунија слика предности и недостатака посматраних софтверских архитектура и виртуелних машина. Једна од њих је свакако време прикупљања смећа (*garbage collection time*) која је директно везана за рад виртуелних машина. Она није укључена у истраживање зато што помоћу коришћених технологија није било могуће мерење вредности ове метрике за апликације које користе GraalVM. Скалирање, као једна од највећих предности микросервисне архитектуре, би такође требало да буде један од критеријума поређења монолита и микросервиса на практичном примеру. Квалитету истраживања би допринели и разноврснији JMeter планови, чије би конфигурације биле специфично прилагођене и намењене појединачним метрикама. Један од недостатака спроведеног истраживања јесте и то што, због раније објашњене некомплатибилности, у апликацијама које користе GraalVM нису примењени Spring Security и Liquibase док у апликацијама које користе JVM јесу, што је у одређеној мери утицало на забележене резултате. Ограниччење које је утицало на евалуацију представља и то што је разматран само један софтверски систем. Поред тога, одлуке о самој декомпозицији монолитног система могу бити другачије у зависности од проблема. Сама апликација се може унапредити претплатом на неки од доступних API-ја који нуде уживо ажуриране податке о фудбалским утакмицама.

7. Литература

- [1] "Software Architecture & Design Introduction", [На мрежи]. Доступно: https://www.tutorialspoint.com/software_architecture_design/introduction.htm [Датум приступа: 27.2.2021.]
- [2] Sam Newman, "Building Microservices", Sebastopol, United States, 01 Feb 2016
- [3] Phill Wittmer, "Monolithic vs Microservices Architecture", [На мрежи]. Доступно: <https://www.tiempodev.com/blog/monolithic-vs-microservices-architecture/> [Датум приступа: 27.2.2021.]
- [4] Синиша Влајић, „ПРОЈЕКТОВАЊЕ СОФТВЕРА (СКРИПТА)“, Београд 2015.
- [5] Синиша Влајић, Душан Савић, Војислав Станојевић, Илија Антовић и Милош Милић, „ПРОЈЕКТОВАЊЕ СОФТВЕРА – НАПРЕДНЕ ЈАВА ТЕХНОЛОГИЈЕ“, Београд, 2008.
- [6] Синиша Влајић, Видојко Ђирић, Душан Савић, „ОСНОВНИ КОНЦЕПТИ ЈАВЕ“, Београд, 2003.
- [7] "How JVM Works – JVM Architecture?", [На мрежи]. Доступно: <https://www.geeksforgeeks.org/jvm-works-jvm-architecture/> [Датум приступа: 9.3.2021.]
- [8] "Compilation and Execution of a Java Program", [На мрежи]. Доступно: <https://www.geeksforgeeks.org/compilation-execution-java-program/> [Датум приступа: 9.3.2021.]
- [9] "Difference between JDK, JRE, and JVM", [На мрежи]. Доступно: <https://www.javatpoint.com/difference-between-jdk-jre-and-jvm> [Датум приступа: 3.3.2021.]
- [10] Oliver B. Fischer, "AN INTRODUCTION TO GRAALVM, ORACLE'S NEW VIRTUAL MACHINE", [На мрежи]. Доступно: <https://jaxlondon.com/blog/an-introduction-to-graalvm-oracles-new-virtual-machine/> [Датум приступа: 13.3.2021.]
- [11] "Introduction to GraalVM", [На мрежи]. Доступно: <https://www.graalvm.org/docs/introduction/> [Датум приступа: 13.3.2021.]
- [12] "Oracle GraalVM Enterprise Edition Technical Brief", [На мрежи]. Доступно: <https://www.oracle.com/a/ocom/docs/graalvm-enterprise-white-paper.pdf> [Датум приступа: 14.3.2021.]
- [13] Josh Long, "Spring Tips: The GraalVM Native Image Builder Feature", [На мрежи]. Доступно: <https://spring.io/blog/2020/04/16/spring-tips-the-graalvm-native-image-builder-feature> [Датум приступа: 21.3.2021.]
- [14] "Native Image Compatibility and Optimization Guide", [На мрежи]. Доступно: <https://www.graalvm.org/reference-manual/native-image/Limitations/> [Датум приступа: 21.3.2021.]
- [15] Andy Clement, Sébastien Deleuze, Filip Hanik, Dave Syer, Esteban Ginez, Jay Bryant, "Spring Native for GraalVM documentation", [На мрежи]. Доступно: <https://repo.spring.io/milestone/org/springframework/experimental/spring-graalvm-native-docs/0.8.3/spring-graalvm-native-docs-0.8.3.zip!/reference/index.html> [Датум приступа: 21.3.2021.]
- [16] Kumar Chandrakant, "Why Choose Spring as Your Java Framework?", [На мрежи]. Доступно: <https://www.baeldung.com/spring-why-to-choose> [Датум приступа: 22.3.2021.]

- [17] Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Alef Arendsen, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abedrabbo, Andy Clement, Dave Syer, Oliver Gierke, Rossen Stoyanchev, Phillip Webb, Rob Winch, Brian Clozel, Stephane Nicoll, Sebastien Deleuze, “Introduction to the Spring Framework”, [На мрежи]. Доступно: <https://docs.spring.io/spring-framework/docs/4.3.20.RELEASE/spring-framework-reference/html/overview.html> [Датум приступа: 22.3.2021.]
- [18] Craig Walls, “Spring in action, fifth edition”, New York, United States, 25 Jan 2019
- [19] Hussein Terek, “Introducing Spring Boot”, [На мрежи]. Доступно: <https://dzone.com/articles/introducing-spring-boot> [Датум приступа: 22.3.2021.]
- [20] “Spring Cloud”, [На мрежи]. Доступно: <https://spring.io/projects/spring-cloud#overview> [Датум приступа: 25.3.2021.]
- [21] “Quick Intro to Spring Cloud Configuration”, [На мрежи]. Доступно: <https://www.baeldung.com/spring-cloud-configuration> [Датум приступа: 26.3.2021.]
- [22] “A Guide to Spring Cloud Netflix – Hystrix”, [На мрежи]. Доступно: <https://www.baeldung.com/spring-cloud-netflix-hystrix> [Датум приступа: 28.3.2021.]
- [23] “Spring Cloud Tutorial - Spring Cloud Gateway Hello World Example”, [На мрежи]. Доступно: <https://www.javainuse.com/spring/cloud-gateway> [Датум приступа: 28.3.2021.]
- [24] Ben Alex, Luke Taylor, Rob Winch, Gunnar Hillert, Joe Grandja, Jay Bryant, Eddú Meléndez, Josh Cummings, Dave Syer, “Spring Security Reference”, [На мрежи]. Доступно: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/> [Датум приступа: 3.4.2021.]
- [25] Amigoscode, “Spring Security | FULL COURSE”, [На мрежи]. Доступно: https://www.youtube.com/watch?v=her_7pa0vrg [Датум приступа: 3.4.2021.]
- [26] Илија Антовић, “Објектно-релационо мапирање”, Факултет организационих наука, 2009.
- [27] “Hibernate - Architecture”, [На мрежи]. Доступно: https://www.tutorialspoint.com/hibernate/hibernate_architecture.htm [Датум приступа: 4.4.2021.]
- [28] “What is SessionFactory in Hibernate?”, [На мрежи]. Доступно: <https://www.java2novice.com/hibernate/session-factory/> [Датум приступа: 4.4.2021.]
- [29] Ramesh Fadatare, “What Is the Difference Between Hibernate and Spring Data JPA?”, [На мрежи]. Доступно: <https://dzone.com/articles/what-is-the-difference-between-hibernate-and-spring-1> [Датум приступа: 4.4.2021.]
- [30] Lars Vogel, “Unit Testing with JUnit - Tutorial”, [На мрежи]. Доступно: <https://www.vogella.com/tutorials/JUnit/article.html> [Датум приступа: 4.4.2021.]
- [31] “React vs Angular vs Vue.js — What to choose in 2020? (updated in 2020)”, [На мрежи]. Доступно: <https://medium.com/techmagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d> [Датум приступа: 5.4.2021.]
- [32] Lukas Marx, “Angular: Everything you need to get started”, [На мрежи]. Доступно: <https://malcoded.com/posts/angular-beginners-guide/> [Датум приступа: 5.4.2021.]

- [33] “Lightweight data shippers”, [На мрежи]. Доступно: <https://www.elastic.co/beats/> [Датум приступа: 6.4.2021.]
- [34] “ELK Stack Tutorial: What is Kibana, Logstash & Elasticsearch?”, [На мрежи]. Доступно: <https://www.guru99.com/elk-stack-tutorial.html> [Датум приступа: 6.4.2021.]
- [35] Aurimas Adomavicius, Rimantas Benetis, Ed Price ,“BUILDING AND DEPLOYING MICROSERVICE APPLICATIONS”, DEVBRIDGE
- [36] Sam Newman, “Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith”, Sebastopol, United States, December 10, 2019.
- [37] Chris Richardson, “Pattern: Monolithic Architecture”, [На мрежи]. Доступно: <https://microservices.io/patterns/monolithic.html> [Датум приступа: 28.5.2021.]
- [38] Синиша Влајић, “Софтверски процес (скрипта)”, Београд, 2016.
- [39] James Lewis, Martin Fowler, “Microservices”, [На мрежи]. Доступно: <https://martinfowler.com/articles/microservices.html> [Датум приступа: 30.5.2021.]
- [40] Lightstep, “Global microservices trends: A survey of development professionals”, April 2018.
- [41] Adrian Colyer, “An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems”, [На мрежи]. Доступно: <https://blog.acolyer.org/2019/05/13/an-open-source-benchmark-suite-for-microservices-and-their-hardware-software-implications-for-cloud-edge-systems/> [Датум приступа: 31.5.2021.]
- [42] Chris Richardson, “Pattern: Microservice Architecture”, [На мрежи]. Доступно: <https://microservices.io/patterns/microservices.html> [Датум приступа: 30.5.2021.]
- [43] Brian Vermeer, “JVM Ecosystem Report 2021”, [На мрежи]. Доступно: <https://res.cloudinary.com/snyk/image/upload/v1623860216/reports/jvm-ecosystem-report-2021.pdf> [Датум приступа: 5.7.2021.]
- [44] “User's Manual”, [На мрежи]. Доступно: <https://jmeter.apache.org/usermanual/glossary.html> [Датум приступа: 30.8.2021.]
- [45] Daniel Berman, “A Metricbeat Tutorial: Getting Started”, [На мрежи]. Доступно: <https://logz.io/blog/metricbeat-tutorial/> [Датум приступа: 30.8.2021.]
- [46] Alexander Kainz, “Microservices vs. Monoliths: An Operational Comparison”, [На мрежи]. Доступно: <https://thenewstack.io/microservices-vs-monoliths-an-operational-comparison/> [Датум приступа: 31.8.2021.]
- [47] Robin Flygare, Anthon Holmqvist, “Performance characteristics between monolithic and microservice-based systems”, [На мрежи]. Доступно: <http://www.diva-portal.org/smash/get/diva2:1119785/FULLTEXT03.pdf> [Датум приступа: 31.8.2021.]
- [48] Noel Welsh, “Benchmarking Web Services using GraalVM Native Image”, [На мрежи]. Доступно: <https://www.inner-product.com/posts/benchmarking-graalvm-native-image/> [Датум приступа: 31.8.2021.]