



Module Code & Module Title
CS5003NI Data Structure and Specialist Programming

Assessment Type
30% Individual Coursework 2

Semester
2023-24 Autumn

Student Name: Sardul Ojha
Project Title: UniMentor
London Met ID: 22067311
College ID: NP01AI4A220030

Assignment Due Date: Friday, May 10, 2024
Assignment Submission Date: Friday, May 10, 2024
Submitted to: Mr. Prithivi Maharjan
Word Count: 15197

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded

Acknowledgement

I extend my sincere gratitude to our Prithvi Sir for granting me the opportunity to undertake this coursework. His guidance and support have been instrumental in navigating the complexities of web application development and Java EE technologies. Under his mentorship, I have gained valuable insights and skills that will undoubtedly shape my future endeavors.

Furthermore, I am indebted to London Metropolitan University and Islington College for providing an enriching academic environment. The resources and facilities offered by these institutions have facilitated my learning journey, allowing me to explore and engage with the subject matter deeply.

Lastly, I would like to express appreciation to my friends, classmates, and teachers for their unwavering support and collaboration throughout this project. Their encouragement, feedback have been invaluable, contributing significantly to the success of this coursework.

Thank you to all those involved for this enriching experience.

Table of Contents

1. Introduction	1
1.1. Business Model.....	1
1.2. Target Audience.....	1
1.3. Unique Selling Proposition.....	1
1.4. Aim	2
1.5. Objectives.....	2
2. User Interface Design	3
2.1. Wireframe.....	3
2.1.1. Home Page (Index)	3
2.1.2. Services Page (For Seekers)	4
2.1.3. My Services Page (For Mentors).....	5
2.1.4. Welcome Page	6
2.1.5. Login Page	7
2.1.6. Register Mentor Page	8
2.1.7. Register Seeker Page	9
2.1.8. Profile Page	10
2.1.9. Add / Update Service Page	11
2.2. Actual design.....	12
2.2.1. Home Page (Index)	12
2.2.2. Services Page (For Seekers)	14
2.2.3. My Services Page (For Mentors).....	14
2.2.4. Welcome Page	15
2.2.5. Login Page	15
2.2.6. Register Mentor Page	16
2.2.7. Register Seeker Page	16
2.2.8. Profile Page	17
2.2.9. Add / Update Service Page	18
3. Class diagram.....	19
3.1. Combined Class Diagram	19
3.2. Individual Class Diagram.....	20

3.2.1. DbController	20
3.2.2. AuthenticationFilter	21
3.2.3. AddServiceServlet	21
3.2.4. IndexServlet.....	21
3.2.5. LoginServlet.....	22
3.2.6. LogoutServlet	22
3.2.7. ModifyServiceServlet.....	22
3.2.8. ProfileServlet	23
3.2.9. RegisterMentorServlet.....	23
3.2.10. RegisterSeekerServlet	23
3.2.11. ServicesServlet.....	24
3.2.12. UpdatePasswordServlet.....	24
3.2.13. LoginModel.....	24
3.2.14. MentorModel.....	25
3.2.15. SeekerModel	26
3.2.16. ServiceModel.....	27
3.2.17. StringUtils	28
3.2.18. ValidationUtils	29
4. Method description	30
4.1. DbController	30
4.1.1. getConnection()	30
4.1.2. registerMentor()	31
4.1.3. registerSeeker().....	32
4.1.4. getLoginInfo()	33
4.1.5. presentInMentor()	34
4.1.6. presentInSeeker().....	35
4.1.7. getAllServiceInfo()	36
4.1.8. getAllMentorServiceInfo()	37
4.1.9. getMentorInfo()	38
4.1.10. getSeekerInfo().....	39
4.1.11. getSeekerImageName()	40
4.1.12. getMentorImageName()	41
4.1.13. getTopMentorsInfo()	42
4.1.14. updateMentor()	43
4.1.15. updateSeeker().....	44
4.1.16. deleteMentor()	45

4.1.17. deleteSeeker()	46
4.1.18. addService()	47
4.1.19. deleteService()	48
4.1.20. updateService()	49
4.1.21. getServiceInfoFromId().....	50
4.1.22. getSearchedServiceInfo().....	51
4.1.23. getSearchedServiceInfoMentor()	52
4.1.24. updateMentorPassword()	53
4.1.25. updateSeekerPassword().....	54
4.2. AuthenticationFilter.....	54
4.2.1. init().....	54
4.2.2. doFilter()	55
4.2.3. destroy()	56
4.3. AddServiceServlet.....	56
4.3.1. AddServiceServlet().....	56
4.3.2. doPost()	57
4.4. IndexServlet.....	58
4.4.1. doGet()	58
4.5. LoginServlet	58
4.5.1. LoginServlet()	58
4.5.2. doPost()	59
4.6. LogoutServlet.....	60
4.6.1. doGet()	60
4.6.2. doPost()	61
4.7. ModifyServiceServlet	61
4.7.1. doGet()	61
4.7.2. doPost()	62
4.7.3. doPut().....	63
4.7.4. doDelete().....	64
4.8. ProfileServlet	65
4.8.1. doGet()	65
4.8.2. doPost()	66
4.8.3. doPut().....	67
4.8.4. doDelete().....	70

4.9. RegisterMentorServlet	71
4.9.1. RegisterMentorServlet()	71
4.9.2. doPost()	71
4.10. RegisterSeekerServlet	73
4.10.1. RegisterSeekerServlet()	73
4.10.2. doPost()	74
4.11. ServicesServlet.....	76
4.11.1. doGet().....	76
4.11.2. doPost()	77
4.12. UpdatePasswordServlet	78
4.12.1. doPost()	78
4.12.2. doPut()	79
4.13. LoginModel	80
4.13.1. LoginModel	80
4.13.2. Getter and Setter methods	81
4.14. MentorModel	82
4.14.1. MentorModel.....	82
4.14.2. Getter and Setter methods	83
4.14.3. getImageUrl().....	87
4.15. SeekerModel	88
4.15.1. SeekerModel	88
4.15.2. Getter and Setter methods	89
4.15.3. getImageUrl().....	92
4.16. ServiceModel	93
4.16.1. ServiceModel.....	93
4.16.2. Getter and Setter methods	94
4.17. ValidationUtils	97
4.17.1. isAlphabetical()	97
4.17.2. isNumeric()	97
4.17.3. isUsername()	97
4.17.4. isPhone()	98
4.17.5. isPassword().....	98
4.17.6. isImage().....	98
4.17.7. isEmail().....	99

5. Database	100
4.1. Assumptions.....	100
4.2. Normalization	100
4.2. Final Entity Relationship Diagram (ERD).....	103
4.3. Database design.....	104
4.4. Table design	104
5. Test cases	107
5.1. Test 1: Test for Register (mentor & seeker).....	107
Test 1.1. : Test for input register details validation	107
Test 1.2. : Test for successful registration	111
5.2. Test 2: Test for Login	116
Test 2.1. : Test for input login details validation	116
Test 2.2. : Test for successful login.....	118
5.3. Test 3: Test for Display	120
Test 3.1. : Test for all services displayed for seekers	120
Test 3.2. : Test for own services displayed for mentors.....	123
5.4. Test 4: Test for Add	125
Test 4.1. : Test for new service being added by mentor	125
Test 4.2. : Test for new added service displayed for seeker.....	129
5.5. Test 5: Test for Update	131
Test 5.1. : Test for selected service being updated for mentor.....	131
Test 5.2. : Test for updated service displayed for seeker	135
5.6. Test 6: Test for Delete	137
Test 6.1. : Test for selected service being deleted for mentor	137
Test 6.2. : Test for deleted service not displayed for seeker	140
5.7. Test 7: Test for Profile (C.R.U.D).....	142
Test 7.1. : Test for new profile being displayed (Create and Read)	142
Test 7.2. : Test for profile being updated (Update)	146
Test 7.3. : Test for profile being deleted.....	151
5.8. Test 8: Test for Password Reset.....	157
Test 8.1. : Test for input validation for password reset	157

Test 8.2. : Test for successful password reset.....	161
5.9. Test 9: Test for Search	168
Test 9.1. : Test for search	168
Test 9.2. : Test for added service being searched	170
Test 9.3. : Test for deleted items not being searched	174
Test 9.4. : Test for updated item being searched	178
5.10. Test 10: Test for navigation (Authentication Filter).....	183
Test 10.1. : Test for access of logged out users	183
Test 10.2. : Test for access for logged in users	187
6. Tools and libraries used	194
3.1. Software Platform Selection.....	194
3.2. Programming PS	197
3.3. Framework and libraries	199
7. Development Process.....	201
7.1. Creating a new dynamic web project.....	201
7.1.1 Project name with location, Apache Tomcat v8.5 and Dynamic module version 3.1	201
7.1.2. Selecting Generate web.xml deployment descriptor.....	202
7.1.3. Creating MVC pattern Packages and Folders.....	202
7.1.4. Creating Folders inside webapp folder.....	202
7.1.5. Adding MySQL Connector jar file in deployment assembly	203
7.1.6. Adding MySQL Connector jar file in java build path	204
7.1.7. Creating DbController Class.....	204
7.1.8. Creating AuthenticationFilter Class	204
7.1.9. Creating Servlet Classes.....	205
7.1.10. Creating Model Class	205
7.1.11. Creating Util Class	205
7.1.12. Creating JSPs	206
7.1.13. Creating Stylesheets (CSS)	206
7.2. Designing Wireframes.....	206
7.3. Website design (Frontend)	207
7.3.1. Home page.....	207
7.3.2. Service page.....	209
7.3.3. Profile page	210

7.3.4. Register page	210
7.3.5. Login page.....	211
7.3.6. Add/Update page.....	211
7.3.7. Welcome page.....	212
7.3.8. Header.....	212
7.3.9. Footer	213
7.4. Backend design.....	213
7.5. Report	215
8. Critical Analysis	216
8.1. Challenges.....	216
8.1.1. Challenge 1: Displaying information from two different tables (i.e. service and mentor tables) in the service page	216
8.1.2. Challenge 2: Updating/Deleteing related table elements from the database.....	218
8.2. Problems Faced	221
8.2.1. Problem Faced 1 : Tomcat Failed to start	221
8.2.2. Problem Faced 2 : Redirection from a servlet to another dyanmic web page	226
8.2.3. Problem Faced 3 : Error when trying to add images.....	231
8.3. Reflection.....	240
9. Conclusion.....	241
10. References.....	242

Table Of Figures

Figure 1: Wireframe of Home page.....	4
Figure 2: Wireframe of Services page	4
Figure 3: Wireframe of my services page	5
Figure 4: Wireframe of Welcome page.....	6
Figure 5: Wireframe of Login Page.....	7
Figure 6: Wireframe of Register Mentor Page	8
Figure 7: Wireframe of register seeker page	9
Figure 8: Wireframe of profile page	10
Figure 9: Wireframe of Add/Update service page	11
Figure 10: Actual Index Page	13
Figure 11: Actual service page	14
Figure 12: Actual my services page	14
Figure 13: Actual welcome page	15
Figure 14: Actual login page	15
Figure 15: Actual register mentor page	16
Figure 16: Actual register seeker page	16
Figure 17: Actual profile page	17
Figure 18: Actual add/update page	18
Figure 19: Overall Class diagram	19
Figure 20: Class diagram of DbController Class.....	20
Figure 21: Class diagram of AuthenticationFilter Class.....	21
Figure 22: Class diagram of AddServiceServlet.....	21
Figure 23: Class diagram of IndexServlet.....	21
Figure 24: Class diagram of LoginServlet.....	22
Figure 25: Class diagram of LogoutServlet.....	22
Figure 26: Class diagram of ModifyServiceServlet.....	22
Figure 27: Class diagram of ProfileServlet	23
Figure 28: Class diagram of RegisterMentorServlet	23
Figure 29: Class diagram of RegisterSeekerServlet	23
Figure 30: Class diagram of ServicesServlet	24
Figure 31: Class diagram of UpdatePasswordServlet	24
Figure 32: Class diagram of LoginModel.....	24
Figure 33: Class diagram of MentorModel	25
Figure 34: Class diagram of SeekerModel.....	26
Figure 35: Class diagram of ServiceModel.....	27
Figure 36: Class diagram of StringUtils.....	28
Figure 37: Class diagram of ValidationUtils	29
Figure 38: Screenshot of getConnction() method	30
Figure 39: Screenshot of registerMentor() method	31
Figure 40: Screenshot of registerSeeker() method.....	32
Figure 41: Screenshot of getAllLoginInfo() method.....	34
Figure 42: Screenshot of presentInMentor() method	34

Figure 43: Screenshot of presentInSeeker() method.....	35
Figure 44: Screenshot of getAllServiceInfo() method	36
Figure 45: Screenshot of getAllMentorServiceInfo() method	37
Figure 46: Screenshot of getMentorInfo() method	38
Figure 47: Screenshot of getSeekerInfo() method	39
Figure 48: Screenshot of getSeekerImageName() method	40
Figure 49: Screenshot of getMentorImageName() method.....	41
Figure 50: Screenshot of getTopMentorsInfo() method	42
Figure 51: Screenshot of updateMentor() method	43
Figure 52: Screenshot of updateSeekerMethod()	44
Figure 53: Screenshot of deleteMentor() method	45
Figure 54: Screenshot of deleteSeeker() method	46
Figure 55: Screenshot of addService() method	47
Figure 56: Screenshot of deleteService() method.....	48
Figure 57: Screenshot of updateService() method	49
Figure 58: Screenshot of getServiceInfoFromId() method	50
Figure 59: Screenshot of getSearchedServiceInfo() method	51
Figure 60: Screenshot of getSearchedServiceInfoMentor() method	52
Figure 61: Screenshot of updateMentorPassword() method	53
Figure 62: Screenshot of updateSeekerPassword() method	54
Figure 63: Screenshot of init() method	54
Figure 64: Screenshot of doFilter() method.....	56
Figure 65: Screenshot of destory() method	56
Figure 66: Screenshot of AddServiceServlet() constructor	56
Figure 67: Screenshot of doPost() method.....	57
Figure 68: Screenshot of doGet() method.....	58
Figure 69: Screenshot of LoginServlet() constructor.....	58
Figure 70: Screenshot of doPost() method.....	60
Figure 71: Screenshot of doGet() method.....	60
Figure 72: Screenshot of doPost() method.....	61
Figure 73: Screenshot of doGet() method.....	61
Figure 74 : Screenshot of doPost() method	62
Figure 75: Screenshot of doPut() method	63
Figure 76: Screenshot of doDelete() method.....	64
Figure 77: Screenshot of doGet() method.....	65
Figure 78: Screenshot of doPost() method	66
Figure 79: Screenshot of doPut() method	69
Figure 80: Screenshot of doDelete() method	70
Figure 81: Screenshot of RegisterMentorServlet() constructor	71
Figure 82: Screenshot of doPost() method.....	72
Figure 83: Screenshot of RegisterSeekerServlet() constructor	73
Figure 84: Screenshot of doPost() method	75
Figure 85: Screenshot of doGet() method.....	77
Figure 86: Screenshot of doPost() method.....	77

Figure 87: Screenshot of doPost() method.....	78
Figure 88: Screenshot of doPut() method	79
Figure 89: Screenshot of LoginModel() constructor	80
Figure 90: Screenshot of getter and setter methods	81
Figure 91: Screenshot of MentorModel() constructor	82
Figure 92: Screenshot of getter and setter methods	86
Figure 93: Screenshot of getImageUrl() method	87
Figure 94: Screenshot of SeekerModel() constructor.....	88
Figure 95: Screenshot of getter and setter methods	91
Figure 96: Screenshot of getImageUrl() method	92
Figure 97: Screenshot of ServiceModel() constructor.....	93
Figure 98: Screenshot of getter and setter methods	96
Figure 99: Screenshot of isAlphabetical() method.....	97
Figure 100: Screenshot of isNumeric() method.....	97
Figure 101: Screenshot of isUsername() method	97
Figure 102: Screenshot of isPhone() method	98
Figure 103: Screenshot of isPassword() method	98
Figure 104: Screenshot of isImage() method	98
Figure 105: Screenshot of isEmail() method	99
Figure 106: Final Entity Relationship Diagram	103
Figure 107: Database and database tables.....	104
Figure 108: Database design	104
Figure 109: Mentor table	105
Figure 110: Seeker Table	105
Figure 111: Service Table	105
Figure 112: Service-details table	106
Figure 113: Screenshot of entering invalid details	108
Figure 114: Screenshot of register button clicked	108
Figure 115: Screenshot of error message displayed	109
Figure 116: Screenshot of details entered in register seeker	109
Figure 117: Screenshot of register button clicked	110
Figure 118: Screenshot of Error message displayed	111
Figure 119: Screenshot of entering invalid details	116
Figure 120: Screenshot of login button clicked	117
Figure 121: Screenshot of error message displayed	117
Figure 122: Screenshot of entering valid login details	118
Figure 123: Screenshot of login button clicked	119
Figure 124: Screenshot of welcome page displayed.....	119
Figure 125: Screenshot of entering seeker login details	120
Figure 126: Screenshot of clicking service button	121
Figure 127: Screenshot of all services in database.....	122
Figure 128: Screenshot of all service displayed in service page.....	122
Figure 129: Screenshot of entering login details as mentor.....	123
Figure 130: Screenshot of clicking my service button in nav.....	124

Figure 131: Screenshot of mento's own service displayed	124
Figure 132: Screenshot of clicking add service button.....	126
Figure 133: Screenshot of entering service details	126
Figure 134: Screenshot of clicking add button	127
Figure 135: Screenshot of new service added.....	128
Figure 136: Screenshot of added service in database.....	128
Figure 137: Screenshot of logging in as seeker.....	129
Figure 138: Screenshot of clicking view services button	130
Figure 139: Screenshot of added service displayed.....	131
Figure 140: Screenshot of clicking updat button	132
Figure 141: Screenshot of details updated	133
Figure 142: Screenshot of update button clicked	133
Figure 143: Screenshot of change in my service.....	134
Figure 144: Screenshot of change in database	135
Figure 145: Screenshot of loging in as a seeker	136
Figure 146: Screenshot of clicking view services.....	136
Figure 147: Screenshot of updated service displayed.....	137
Figure 148: Screenshot of delete button clicked	138
Figure 149: Screenshot of ok button clicked	139
Figure 150: Screenshot of deleted service not displayed	139
Figure 151: Screenshot of deleted service remorved from database	140
Figure 152: Screenshot of logging in as seeker.....	141
Figure 153: Screenshot of deleted service not displayed for seeker.....	142
Figure 154: Screenshot of logging in as a user	143
Figure 155: Screenshot of clicking profile button	144
Figure 156: Screenshot of profile information displayed	145
Figure 157: Screenshot of profile information in database	145
Figure 158: Screenshot of clicking profile button on navbar	146
Figure 159: Screenshot of profile information displayed	147
Figure 160: Screenshot of profile information updated	148
Figure 161: Screenshot of update profile button clicked	148
Figure 162: Screenshot of again clicking on profile button	149
Figure 163: Screenshot of updated information displayed	150
Figure 164: Screenshot of information updated in database	150
Figure 165: Screenshot of clicking profile buttons.....	152
Figure 166: Screenshot of profile information displayed	152
Figure 167: Screenshot of clicking delete button	153
Figure 168: Screenshot of clicking yes for confirmation.....	154
Figure 169: Screenshot of redirection to index page logged out.....	154
Figure 170: Screenshot of account removed from database	155
Figure 171: Screenshot of entering login information of deleted account.....	156
Figure 172: Screenshot of error message displayed	157
Figure 173: Screenshot of clicking profile button	158
Figure 174: Screenshot of profile displayed	159

Figure 175: Screenshot of entering invalid in password reset	159
Figure 176: Screenshot of update password button clicked	160
Figure 177: Screenshot of clicking profile button	162
Figure 178: Screenshot of profile information displayed	162
Figure 179: Screenshot of entering correct information in password fields	163
Figure 180: Screenshot of clicking update password button	164
Figure 181: Screenshot of redirection to welcome page	164
Figure 182: Screenshot of clicking logout button	165
Figure 183: Screenshot of entering old password.....	166
Figure 184: Screenshot of error message displayed	166
Figure 185: Screenshot of entering correct password.....	167
Figure 186: Screenshot of login success	168
Figure 187: Screenshot of entering search value.....	169
Figure 188: Screenshot of search button clicked	169
Figure 189: Screenshot of search results displayed	170
Figure 190: Screenshot of clicking add button	171
Figure 191: Screenshot of addind new service.....	172
Figure 192: Screenshot of entering added service name.....	172
Figure 193: Screenshot of search button clicked	173
Figure 194: Screenshot of added service displayed.....	174
Figure 195: Screenshot of clicking delete button	175
Figure 196: Screenshot of clicking ok button	176
Figure 197: Screenshot of entering deleted service title	176
Figure 198: Screenshot of search button clicked	177
Figure 199: Screenshot of no service found	178
Figure 200: Screenshot of clicking update button	179
Figure 201: Screenshot of updating title	180
Figure 202: Screenshot of entering old title in searchbox.....	180
Figure 203: Screenshot of old service title not displayed	181
Figure 204: Screenshot of entering updated title.....	182
Figure 205: Screenshot of updated service displayed	182
Figure 206: Screenshot of clicking services in nav bar	184
Figure 207: Screenshot of login page displayed	184
Figure 208: Screenshot of giving profile page url	185
Figure 209: Screenshot of login page displayed	186
Figure 210: Screenshot of giving welcome page url	186
Figure 211: Screenshot of login page displayed	187
Figure 212: Screenshot of logging in as user	188
Figure 213: Screenshot of welcome page displayed.....	189
Figure 214: Screenshot of giving login page url	189
Figure 215: Screenshot of welcome page displayed.....	190
Figure 216: Screenshot of giving register mentor page url	191
Figure 217: Screenshot of welcome page displayed.....	191
Figure 218: Screenshot of giving register seeker page url	192

Figure 219: Screenshot of welcome page displayed.....	193
Figure 220: Logo of eclipse ide	194
Figure 221: Logo of tomcat server.....	194
Figure 222: Logo of XAMPP.....	195
Figure 223: Logo of balsamiq	196
Figure 224: Logo of MS Word	196
Figure 225: Logo of draw.io	197
Figure 226: Logo of Java	197
Figure 227: HTML, CSS AND JS LOGO	198
Figure 228: Logo of MySQL.....	198
Figure 229: Figure of servlets and JSP	199
Figure 230: Diagram of JDBC	200
Figure 231: Screenshot of giving project name and selecting server	201
Figure 232: Screenshot of generating webxml description	202
Figure 233: Screenshot of creating packages.....	202
Figure 234: Screenshot of creating folders	202
Figure 235: Screenshot of adding mysql jar in deployment assembly	203
Figure 236: Screenshot of adding mysql jar in java build path	204
Figure 237: Screenshot of creating DbController class	204
Figure 238: Screenshot of creating authentication filter class.....	204
Figure 239: Screenshot of creating servlet classes.....	205
Figure 240: Screenshot of creating model class.....	205
Figure 241: Screenshot of creating util classes	205
Figure 242: Screenshot of creating jsp	206
Figure 243: Screenshot of creating css files.....	206
Figure 244: Screenshot of designing wireframes	207
Figure 245: Screenshot of backend development	214
Figure 246: Screenshot of implementing hashmap	216
Figure 247: Screenshot of getting solution from stack overflow.....	217
Figure 248: Screenshot of getting hashmap values in jsp.....	218
Figure 249: Screenshot of when delete button clicked	219
Figure 250: Screenshot of blank page displayed	219
Figure 251: Screenshot of displayed error	220
Figure 252: Screenshot of updating/deleting manally from database.....	220
Figure 253: Screenshot of changing key type to cascade	221
Figure 254: Screenshot of tomcat failed to start.....	222
Figure 255: Screenshot of cleaning tomcat	223
Figure 256: Screenshot of restarting tomcat.....	223
Figure 257: Screenshot of deleting other projects	224
Figure 258: Screenshot of running the program	225
Figure 259: Screenshot of tomcat started and program running	225
Figure 260: Screenshot of redirection code	226
Figure 261: Screenshot of entering invalid in reset password	227
Figure 262: Screenshot of blank page shown (error).....	227

Figure 263: Screenshot of changed code	228
Figure 264: Screenshot of changed code in profile servlet	229
Figure 265: Screenshot of entering invalid input again in reset password	230
Figure 266: Screenshot of error message displayed (code working).....	230
Figure 267: Screenshot of entering valid registration details	231
Figure 268: Screenshot of error page	232
Figure 269: Screenshot of adding breakpoints for debugging	233
Figure 270: Screenshot of running debugger.....	233
Figure 271: Screenshot of registering again	234
Figure 272: Screenshot of program stopped at breakpoint.....	235
Figure 273: Screenshot of null values returned	236
Figure 274: Screenshot of hint in stack overflow	236
Figure 275: Screenshot of corrected/missing code in github	237
Figure 276: Screenshot of missing code added in servlet	238
Figure 277: Screenshot of registering again	238
Figure 278: Screenshot of register success (problem solved).....	239

Table Of Tables

Table 1: Test table for register details validation	107
Table 2: Test table for success reigstration	111
Table 3: Test table for login details validation	116
Table 4: Test table for login success	118
Table 5: Test table for service display to seekers	120
Table 6: Test table for service display to mentors	123
Table 7: Test table for new service added	125
Table 8: Test table for added service displayed	129
Table 9: Test table for selected service update.....	131
Table 10: Test table for updated service display	135
Table 11: Test table for selected service delete	137
Table 12: Test table for deleted service not displayed for seeker.....	140
Table 13: Test table for profile display	142
Table 14: Test table for profile update	146
Table 15: Test table for profile delete.....	151
Table 16: Test table for validation for password reset.....	157
Table 17: Test table for successful password reset.....	161
Table 18: Test table for search	168
Table 19: : Test table for added service search	170
Table 20: Test table for deleted service search.....	174
Table 21: Test table for updated service search.....	179
Table 22: Test table for page access to logged out users	183
Table 23: Test table for page access to logged in users	187

1. Introduction

Welcome to our platform dedicated to empowering students globally. We serve as a nexus between university seekers and mentors, facilitating invaluable guidance for those embarking on their journey to study abroad. With personalized connections, seekers gain access to tailored advice on universities, scholarships, and more, while mentors seize the opportunity to share their expertise and earn income. Our mission is to simplify the application process and foster a diverse exchange of perspectives, transcending geographical barriers. Join us in revolutionizing the way students navigate the path to higher education, making dreams of studying abroad a tangible reality for all.

1.1. Business Model

The platform works as a two-sided marketplace connecting university seekers (students looking for guidance) with mentors (university students/ alumni offering for services). The revenue generation mostly comes from transaction fee, advertising, and partnerships with universities.

1.2. Target Audience

❖ University Searching Students:

Undergraduate and graduate students seeking guidance, advice, and mentorship regarding university selection, scholarships, and other related matters.

❖ University Students/Alumni:

Current students or alumni are willing to earn some money in exchange for their guidance, mentorship, and advice to prospective students.

1.3. Unique Selling Proposition

- This platform offers direct connection between searching students and university students.
- It provides seekers the ability to search mentors and services based on specific criteria such as university, price, location, and so on according to their preferences.

- This platform provides a convenient way for searching students to connect with university students from around the world, eliminating geographical barriers and offering a diverse range of perspectives.

1.4. Aim

The main aim of the platform is to simplify the university application process for students seeking to study abroad by connecting them with experienced university students or alumni who offer guidance and support.

1.5. Objectives

- To connect university seekers with students(mentors) from their dream university.
- To simplify the university application process.
- To offer guidance on universities, scholarships, and benefits.
- To provide access to a wide range of services offered by students from various universities worldwide.
- To provide a source of income for unemployed university students abroad.

2. User Interface Design

2.1. Wireframe

2.1.1. Home Page (Index)

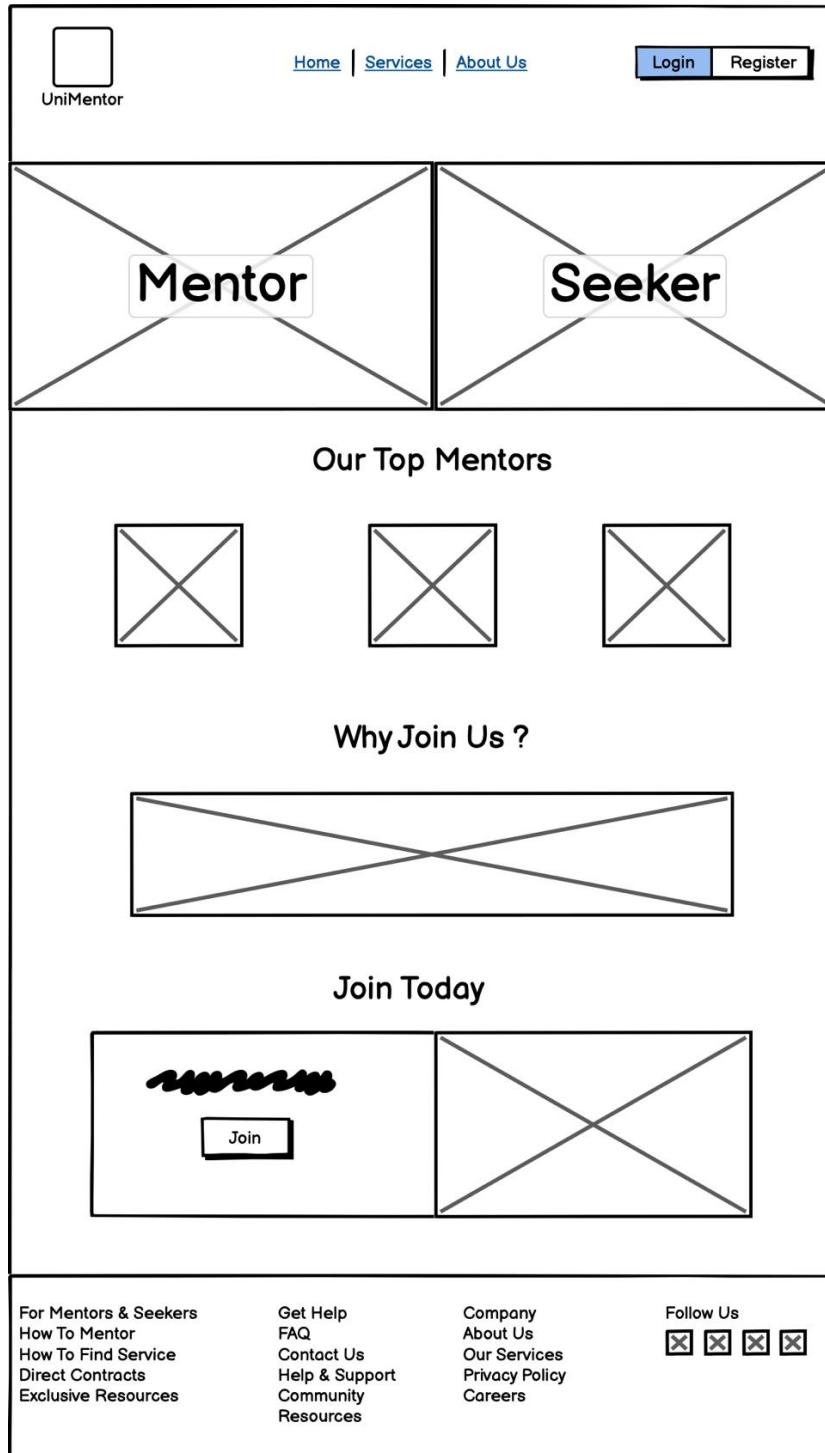
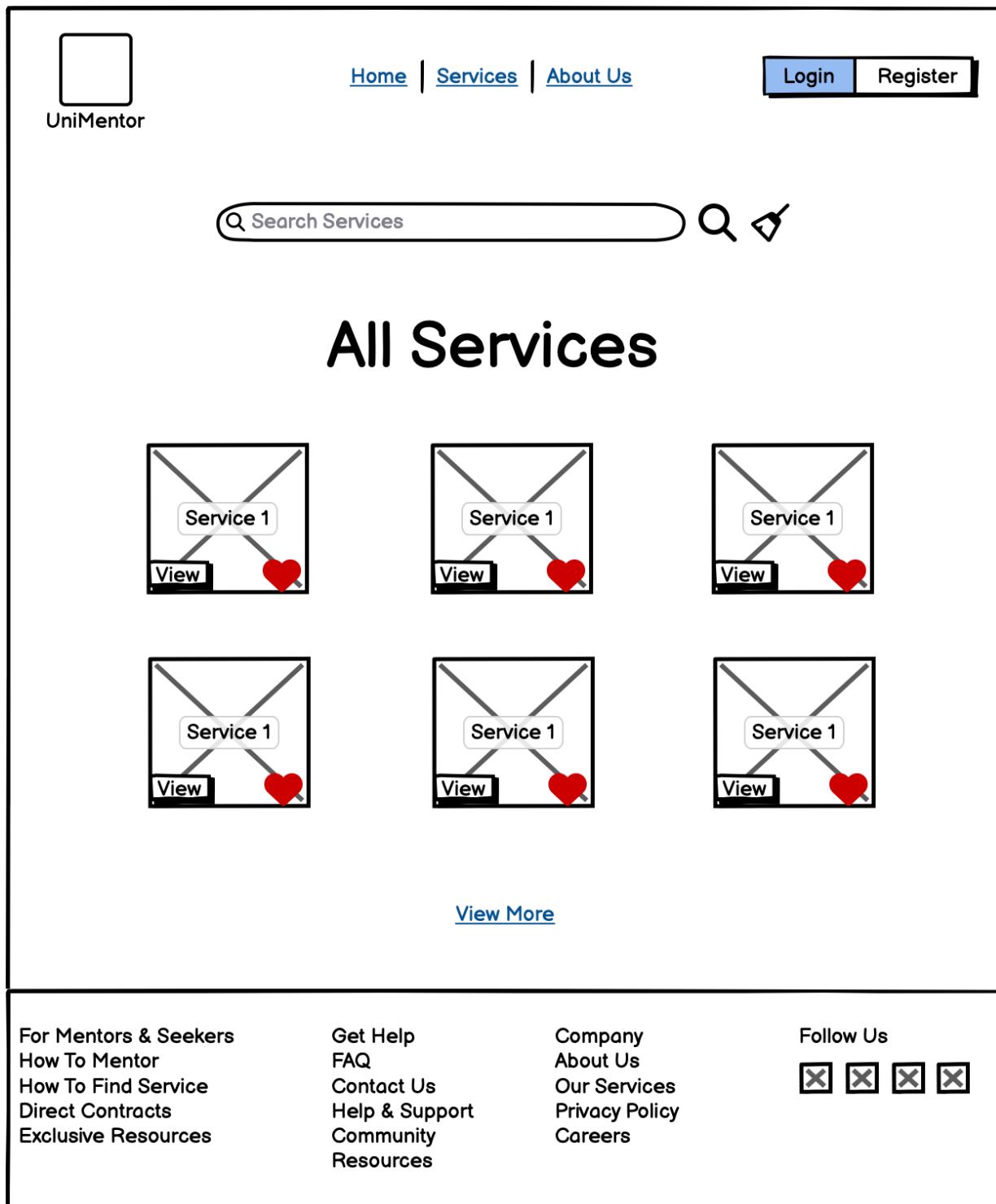


Figure 1: Wireframe of Home page

2.1.2. Services Page (For Seekers)

*Figure 2: Wireframe of Services page*

2.1.3. My Services Page (For Mentors)

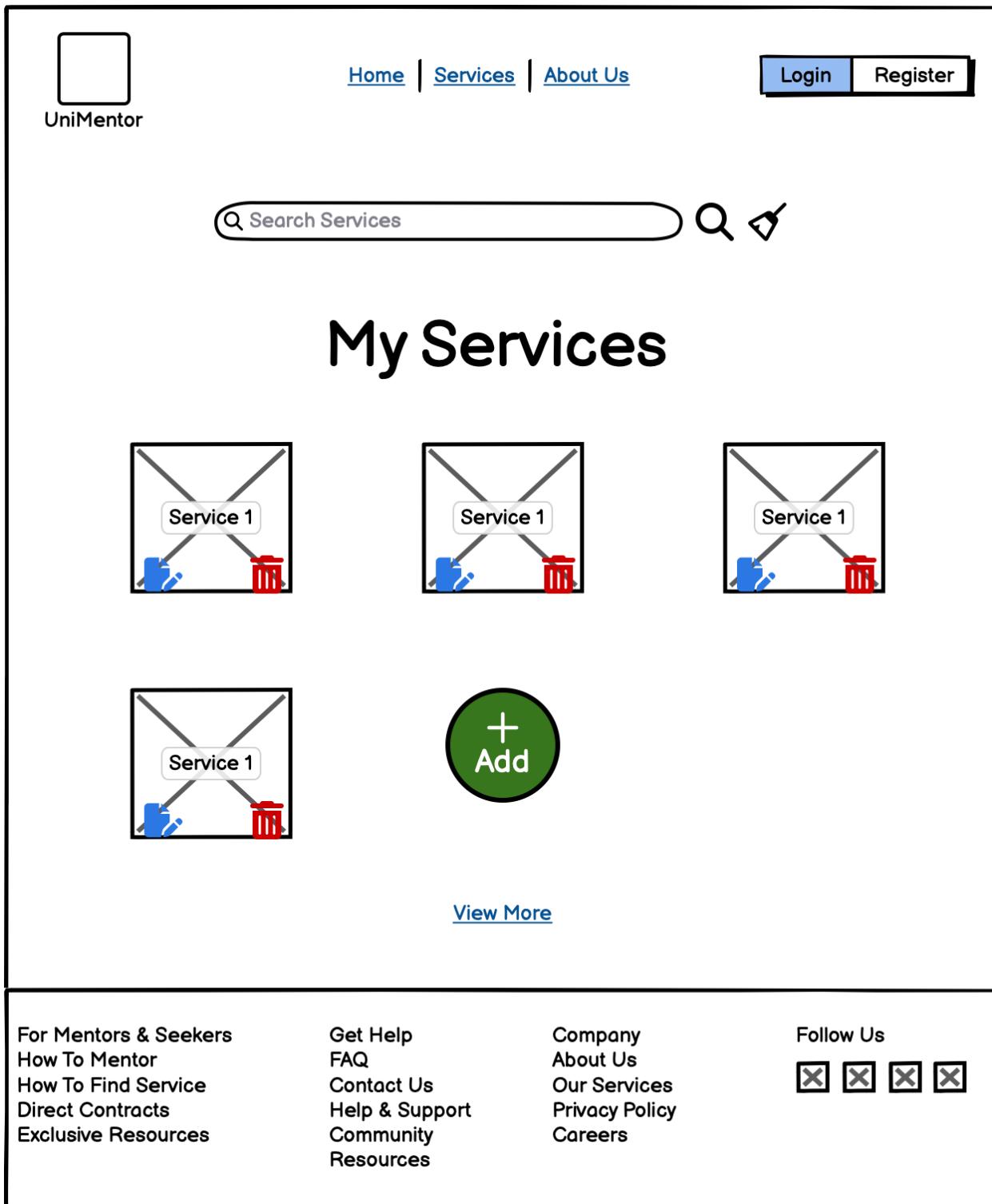


Figure 3: Wireframe of my services page

2.1.4. Welcome Page

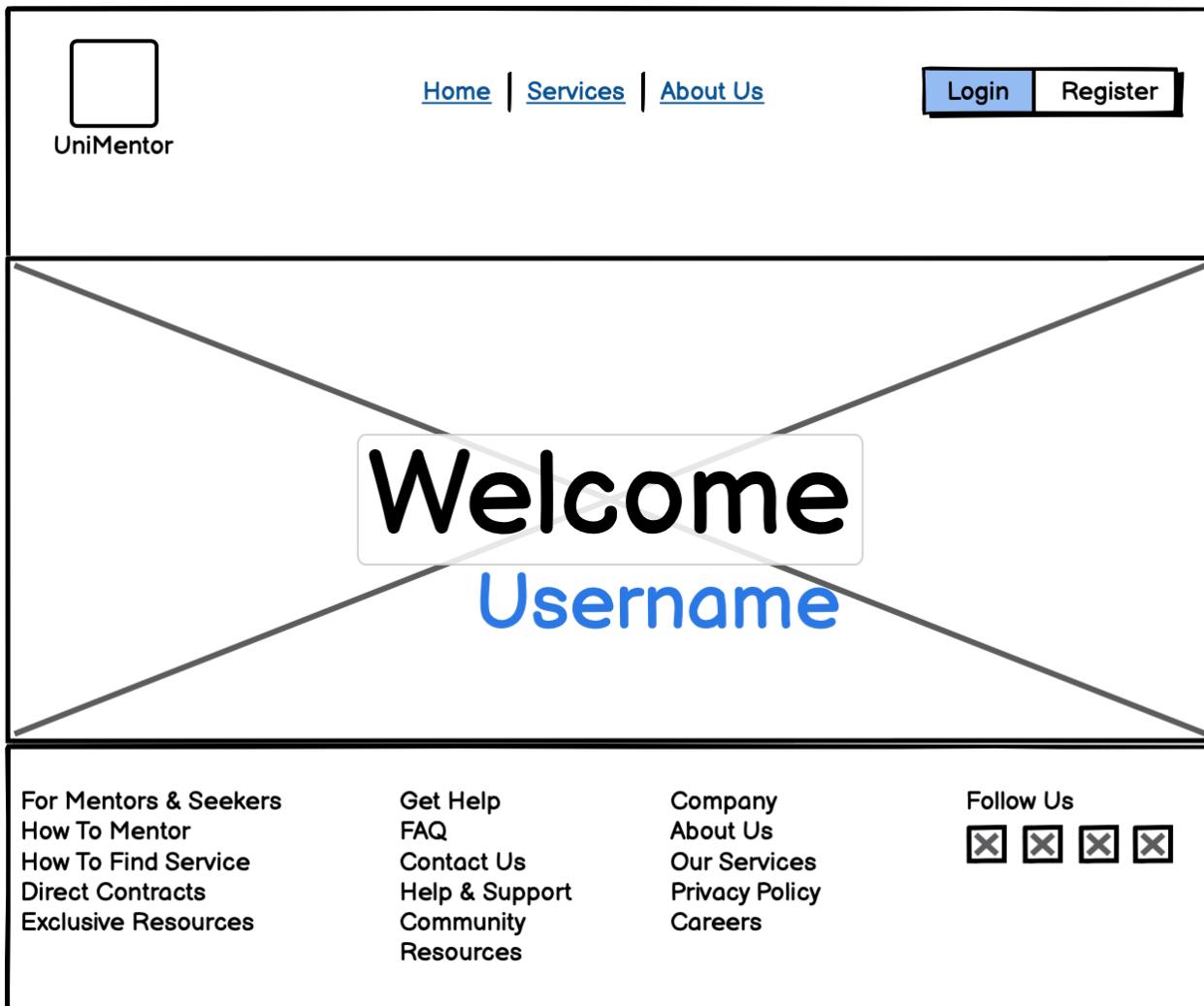


Figure 4: Wireframe of Welcome page

2.1.5. Login Page

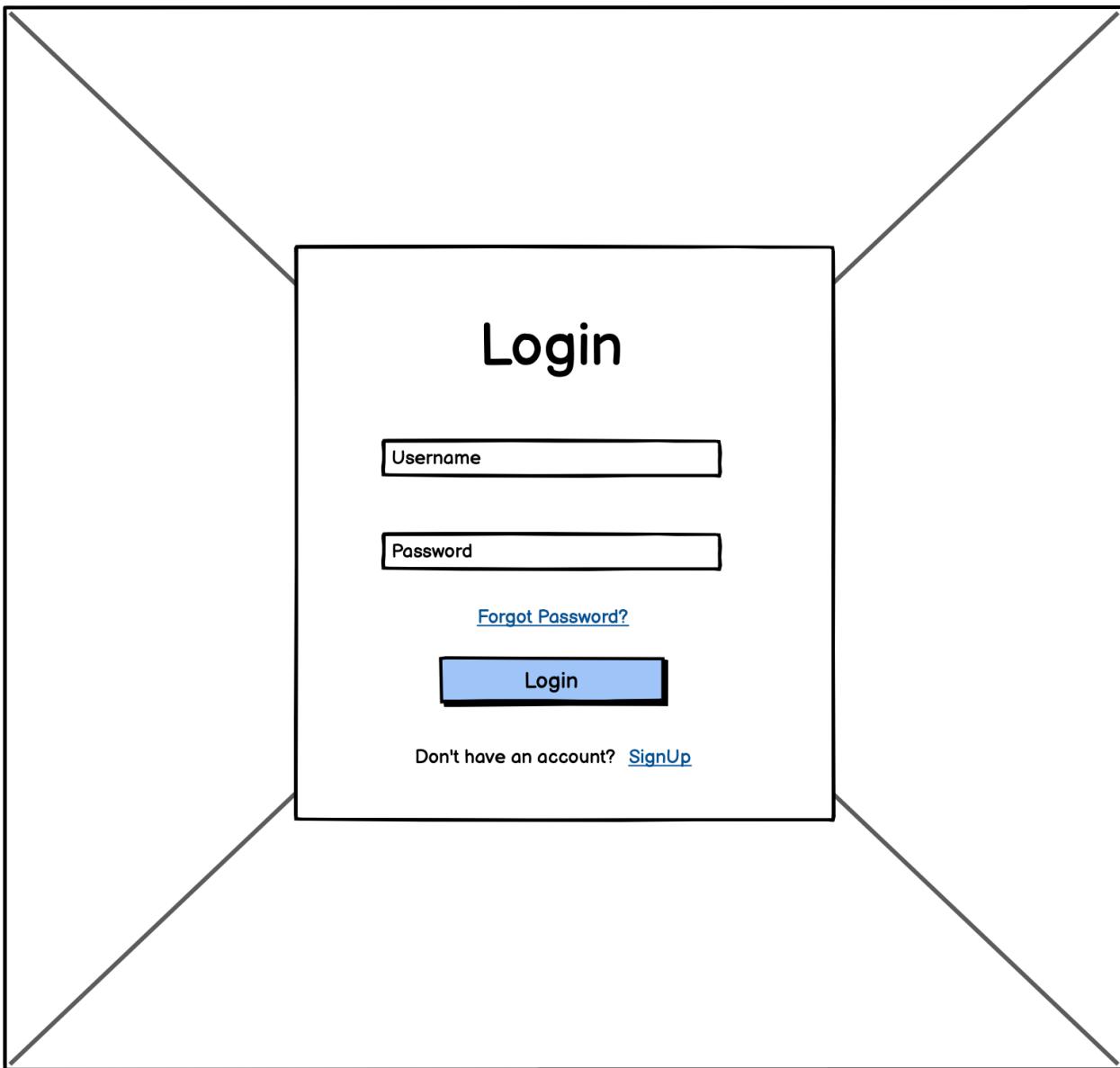


Figure 5: Wireframe of Login Page

2.1.6. Register Mentor Page

The wireframe illustrates the 'Register as a Mentor' page. At the top right is a link to 'Register as a Seeker ->'. The page is divided into two main sections: 'Personal Details' and 'University Details'. Under 'Personal Details', there are fields for Name, Username, Email, Phone, Password, and Confirm Password. Under 'University Details', there are fields for University Name, University Country, Major, Tuition Fee, Scholarship, and Profile Photo. A large blue 'Register' button is positioned at the bottom left of the form area. Below the form, a note says 'Already have an account? [Login](#)'.

Figure 6: Wireframe of Register Mentor Page

2.1.7. Register Seeker Page

The wireframe illustrates the 'Register as a Seeker' page. At the top right is a link 'Register as a Mentor ->'. The page is titled 'Register as a Seeker'. It contains a section labeled 'Personal Details' with the following fields:

- Name (text input)
- Username (text input)
- Email (text input)
- Phone (text input)
- Password (text input)
- Confirm Password (text input)
- Location (text input)
- Education Level (dropdown menu)
- Profile Photo (file input)

A blue 'Register' button is located below the fields. At the bottom left, there is a link 'Already have an account? [Login](#)'.

Figure 7: Wireframe of register seeker page

2.1.8. Profile Page

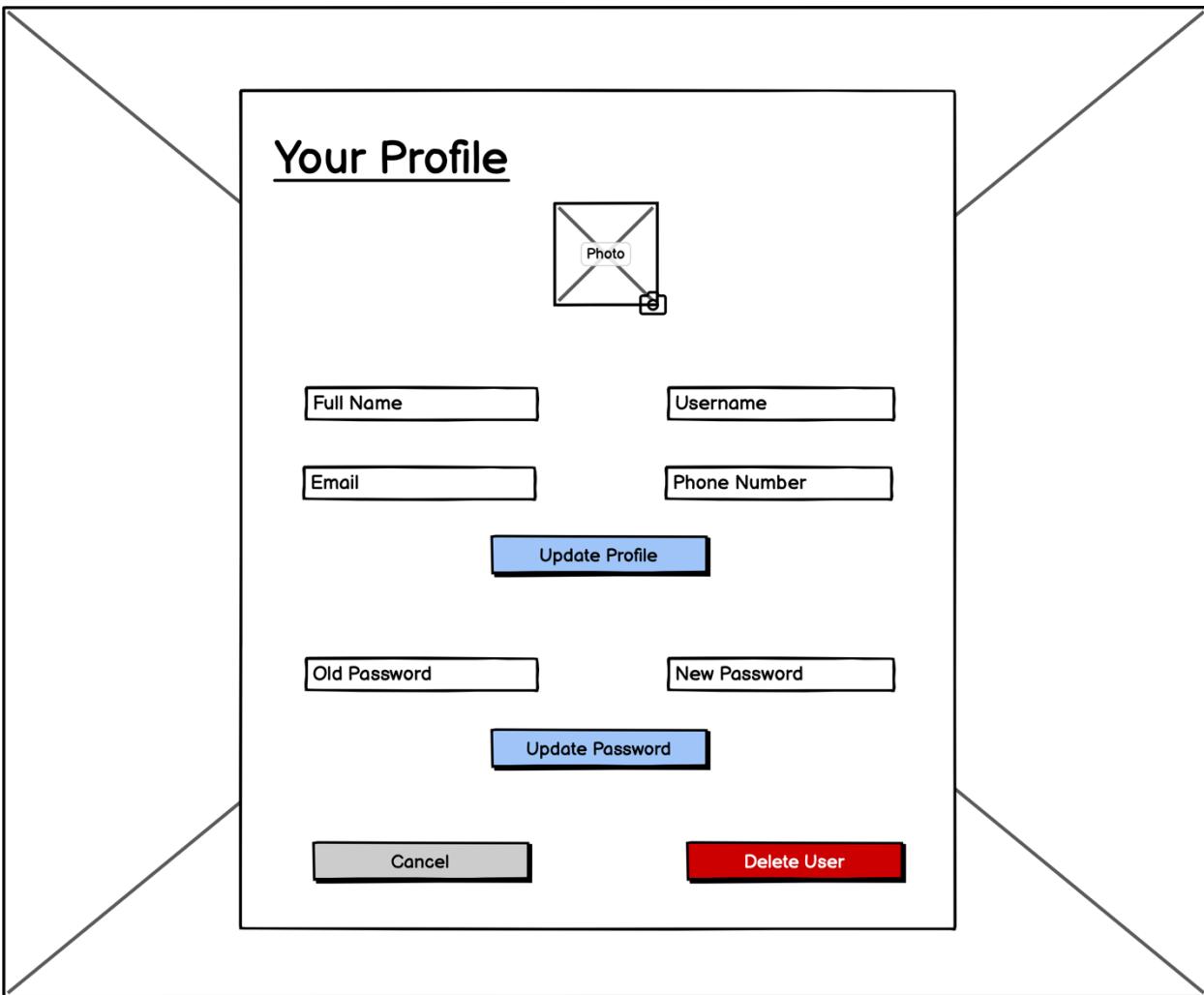


Figure 8: Wireframe of profile page

2.1.9. Add / Update Service Page

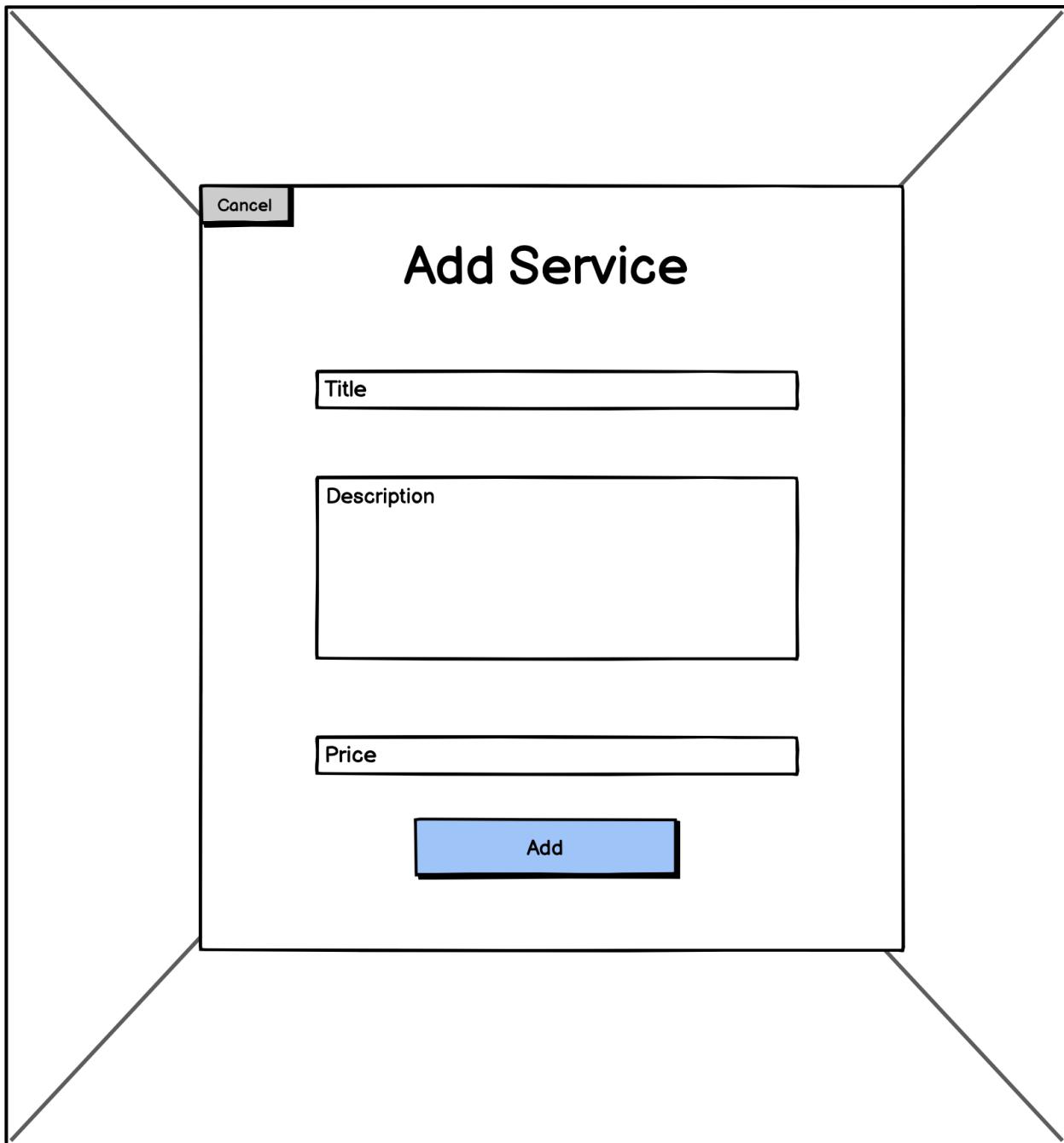
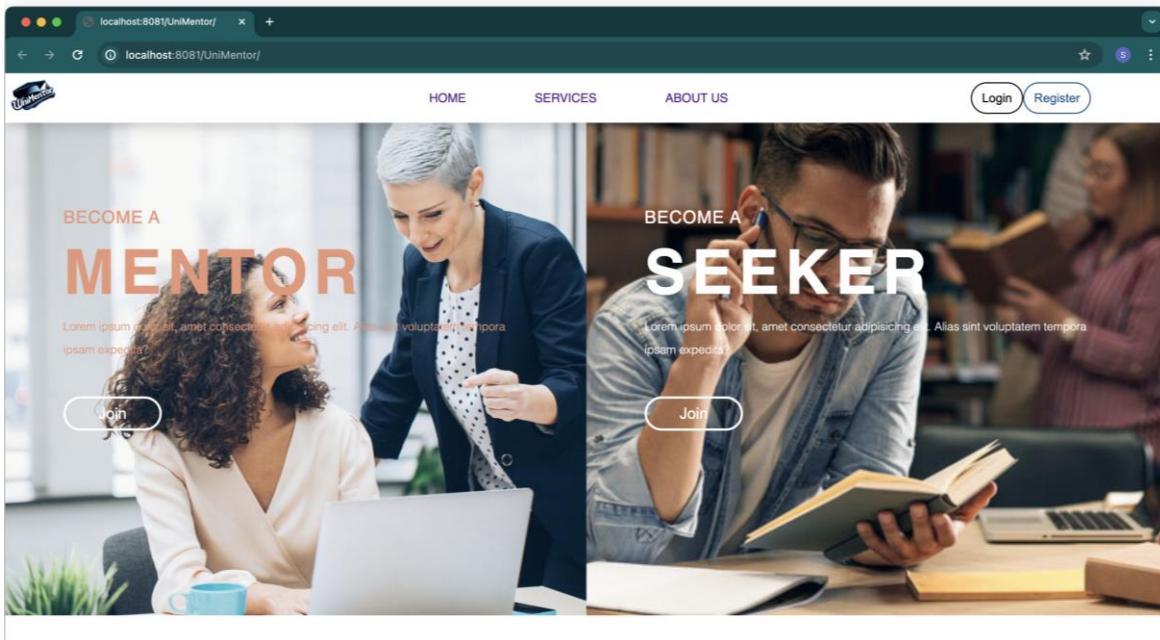


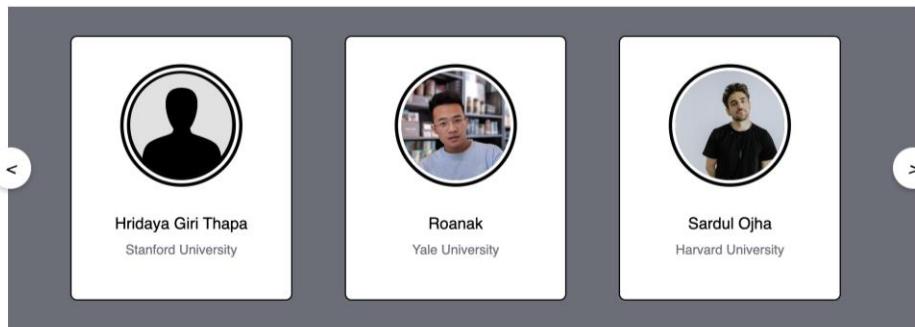
Figure 9: Wireframe of Add/Update service page

2.2. Actual design

2.2.1. Home Page (Index)



OUR TOP MENTORS



WHY JOIN US ?**Personal Trainer**

- Unlock your potential with our expert Personal Trainers.

Practice Sessions

- Elevate your creativity with practice sessions.

Good Management

- Supportive management, for your career success.

JOIN TODAY

**Abroad Study Services at your
Fingertips**

[Join Now](#)**For Mentors & Seekers**

- How To Mentor
- How To Find Service
- Direct Contracts
- Exclusive Resources

Get Help

- FAQ
- Contact Us
- Help & Support
- Community
- Resources

Company

- About Us
- Our Services
- Privacy Policy
- Careers

Follow Us

Figure 10: Actual Index Page

2.2.2. Services Page (For Seekers)

The screenshot shows the 'All Services' page. It features three service cards:

- Interview Preparation Package** by Hridaya Giri Thapa (Stanford University, 80% Scholarship). Price: \$150, Rating: 3.0 stars, 289 Users. Includes a 'View Details' button and a red heart icon.
- Full Application** by Sardul Ojha (Harvard University, 100% Scholarship). Price: \$499, Rating: 4.3 stars, 19 Users. Includes a 'View Details' button and a red heart icon.
- Recommendation Letter Assistance** by Hridaya Giri Thapa (Stanford University, 80% Scholarship). Price: \$99, Rating: 0.0 stars, 0 Users. Includes a 'View Details' button and a red heart icon.

Figure 11: Actual service page

2.2.3. My Services Page (For Mentors)

The screenshot shows the 'My Services' page. It features two service cards and an 'Add Service' button:

- Interview Preparation Package** by Hridaya Giri Thapa (Stanford University, 80% Scholarship). Price: \$150, Rating: 3.0 stars, 289 Users. Includes a pencil icon and a trash bin icon.
- Recommendation Letter Assistance** by Hridaya Giri Thapa (Stanford University, 80% Scholarship). Price: \$99, Rating: 0.0 stars, 0 Users. Includes a pencil icon and a trash bin icon.

Add Service

Figure 12: Actual my services page

2.2.4. Welcome Page

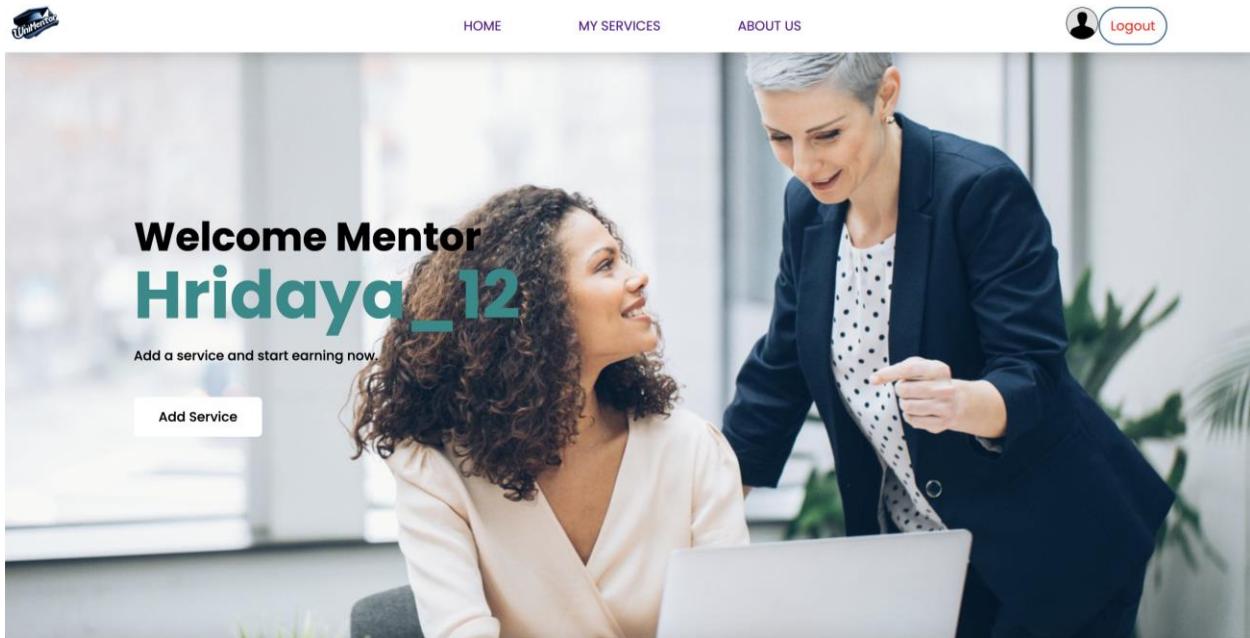


Figure 13: Actual welcome page

2.2.5. Login Page

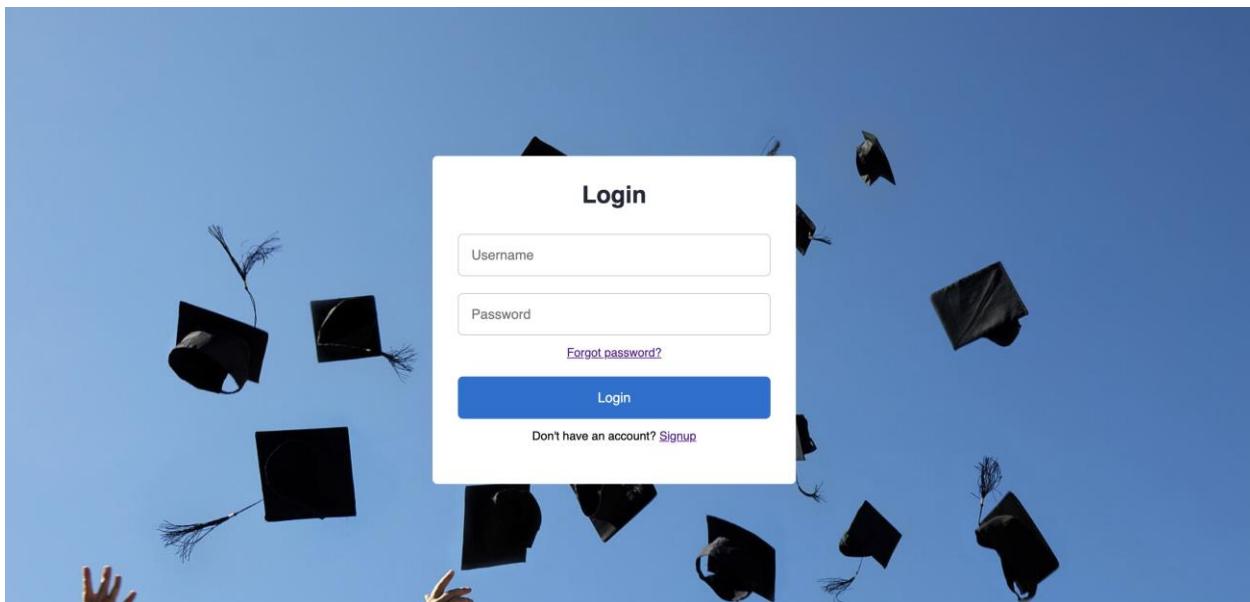
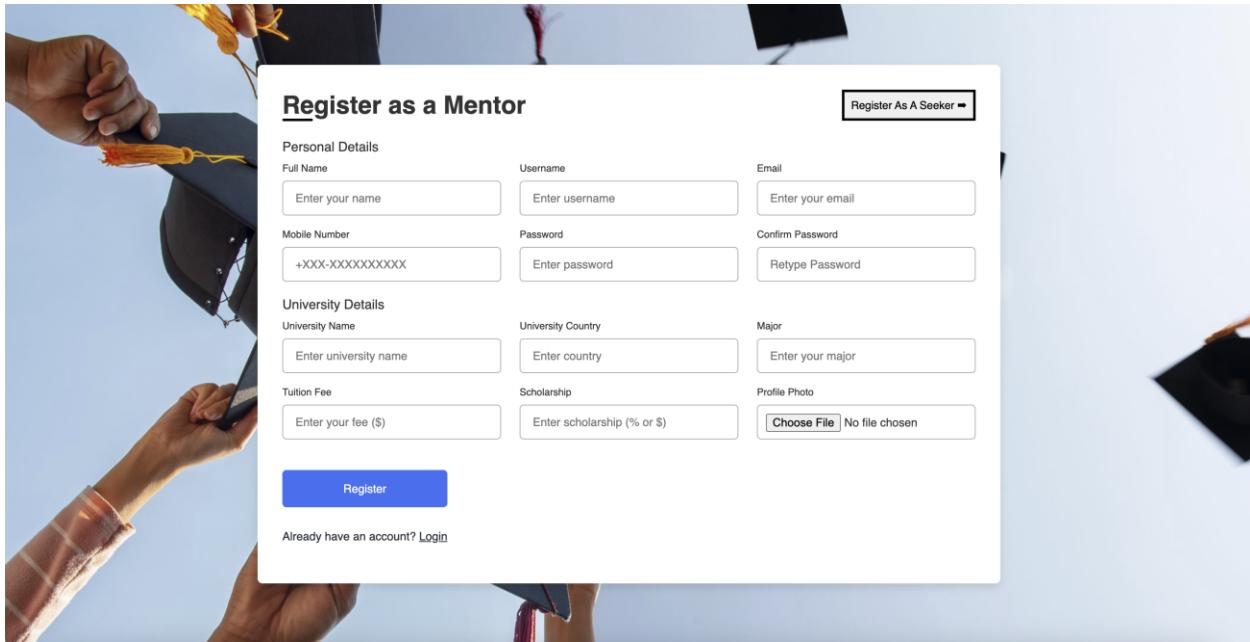


Figure 14: Actual login page

2.2.6. Register Mentor Page



Register as a Mentor

Personal Details

Full Name Enter your name	Username Enter username	Email Enter your email
Mobile Number +XXXX-XXXXXX	Password Enter password	Confirm Password Retype Password

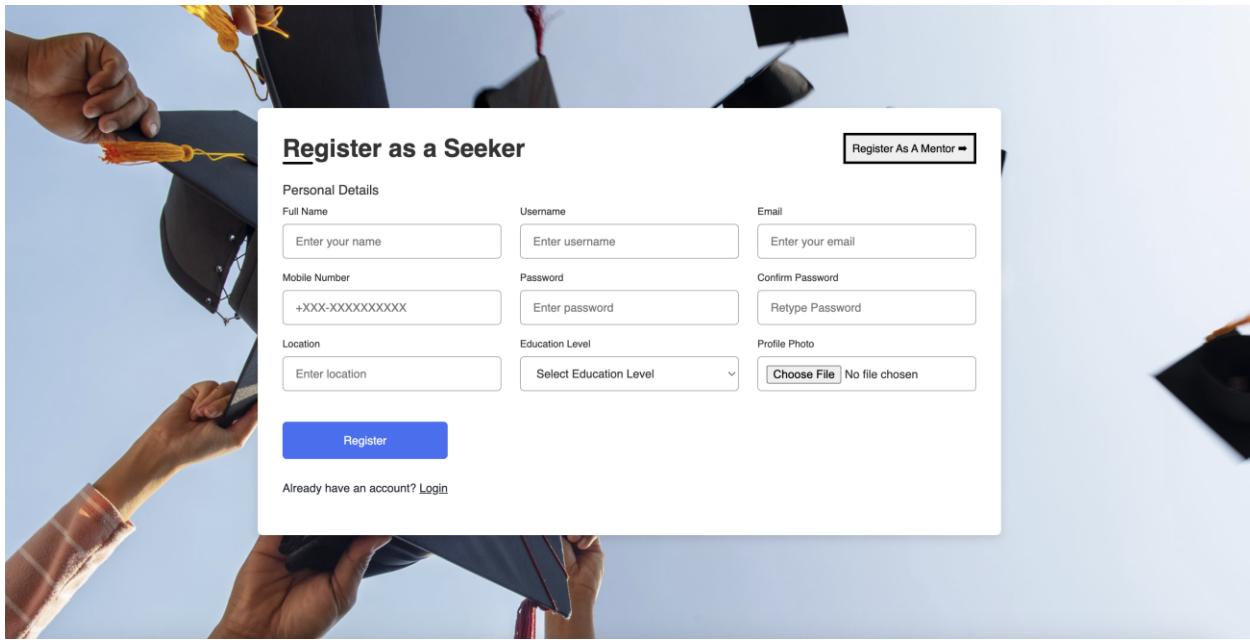
University Details

University Name Enter university name	University Country Enter country	Major Enter your major
Tuition Fee Enter your fee (\$)	Scholarship Enter scholarship (% or \$)	Profile Photo <input type="button" value="Choose File"/> No file chosen

Buttons: Register, Already have an account? [Login](#)

Figure 15: Actual register mentor page

2.2.7. Register Seeker Page



Register as a Seeker

Personal Details

Full Name Enter your name	Username Enter username	Email Enter your email
Mobile Number +XXXX-XXXXXX	Password Enter password	Confirm Password Retype Password

Location Enter location	Education Level Select Education Level	Profile Photo <input type="button" value="Choose File"/> No file chosen
----------------------------	---	--

Buttons: Register, Already have an account? [Login](#)

Figure 16: Actual register seeker page

2.2.8. Profile Page

The screenshot shows a user profile page titled "Your Profile". At the top center is a placeholder circular profile picture with a black silhouette of a person. To the right of the picture is a small teal circular icon containing a white camera symbol. Below the profile area are four input fields arranged in a 2x2 grid:

Full Name Hridaya Giri Thapa	Username Hridaya_12
Email hir@hotmail.com	Phone Number +913-8888888888

Below these fields is a large teal rectangular button labeled "Update Profile". A horizontal line separates this section from the password update section below.

The password update section contains two input fields:

Old Password Enter previous password	New Password Enter new password
---	------------------------------------

Below these fields is another large teal rectangular button labeled "Update Password". A horizontal line separates this section from the bottom buttons.

At the bottom left is a grey rectangular button labeled "Cancel". At the bottom right is a pink rectangular button labeled "Delete Account".

Figure 17: Actual profile page

2.2.9. Add / Update Service Page

The screenshot shows a mobile-style application interface titled "Add Service". At the top left is a "Cancel" button with a back arrow icon. The main title "Add Service" is centered at the top in a large, bold, black font. Below the title are three input fields: "TITLE" (with an empty text input box), "DESCRIPTION" (with an empty text input box), and "PRICE" (with an empty text input box). At the bottom is a teal-colored "Add" button with white text.

Figure 18: Actual add/update page

3. Class diagram

3.1. Combined Class Diagram

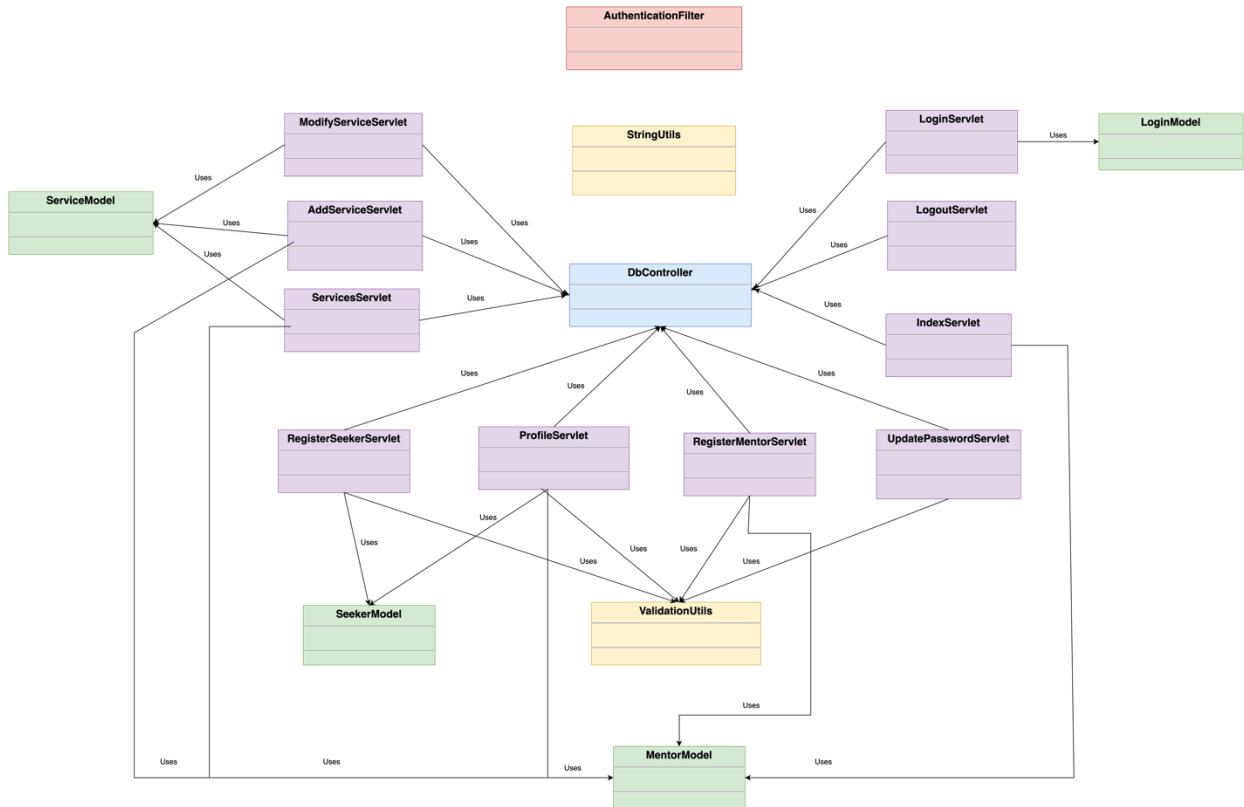


Figure 19: Overall Class diagram

3.2. Individual Class Diagram

3.2.1. DbController

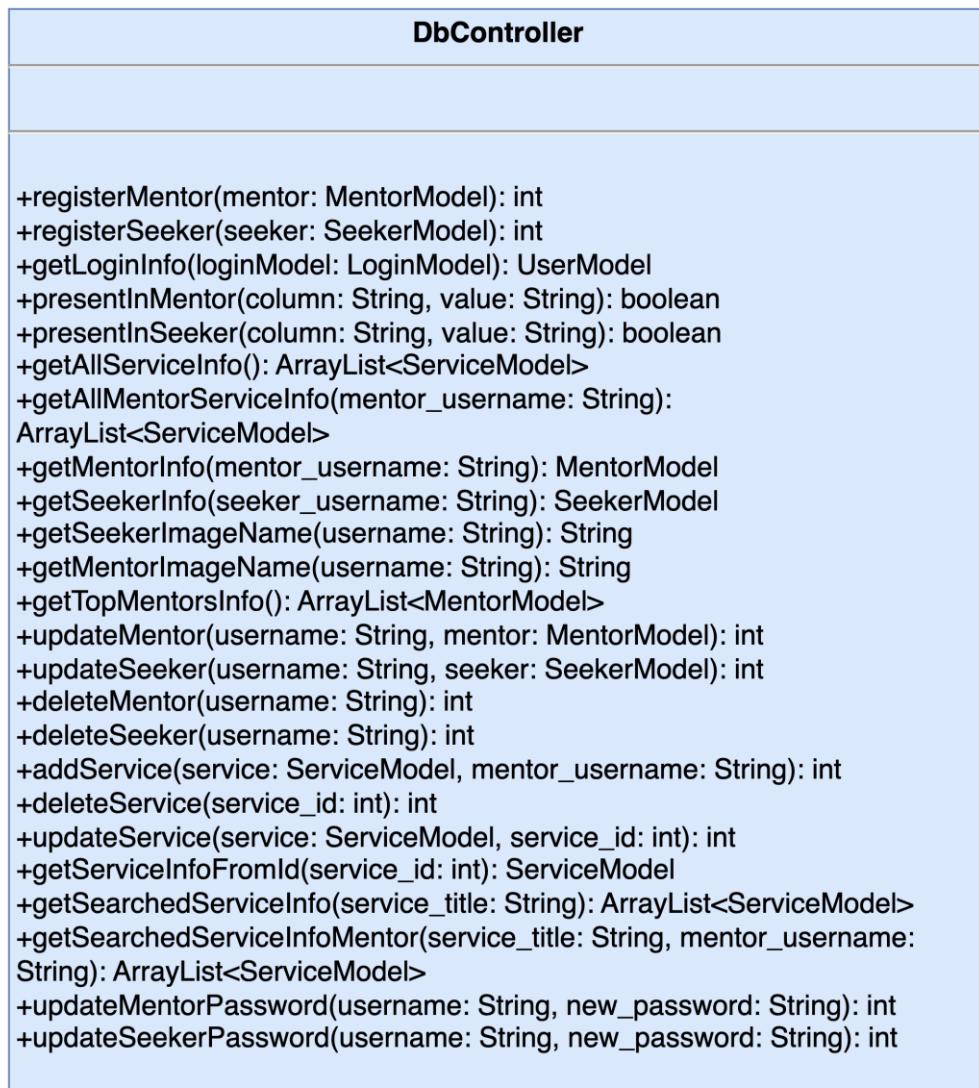


Figure 20: Class diagram of DbController Class

3.2.2. AuthenticationFilter

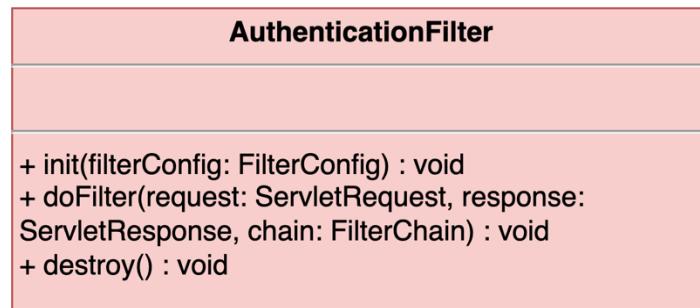


Figure 21: Class diagram of AuthenticationFilter Class

3.2.3. AddServiceServlet



Figure 22: Class diagram of AddServiceServlet

3.2.4. IndexServlet

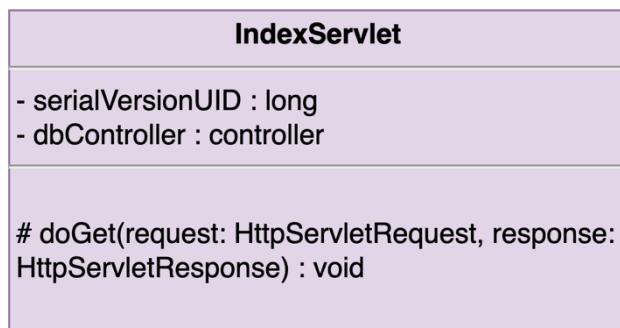


Figure 23: Class diagram of IndexServlet

3.2.5. LoginServlet

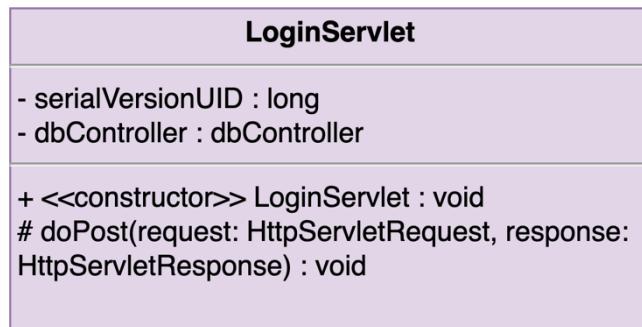


Figure 24: Class diagram of LoginServlet

3.2.6. LogoutServlet

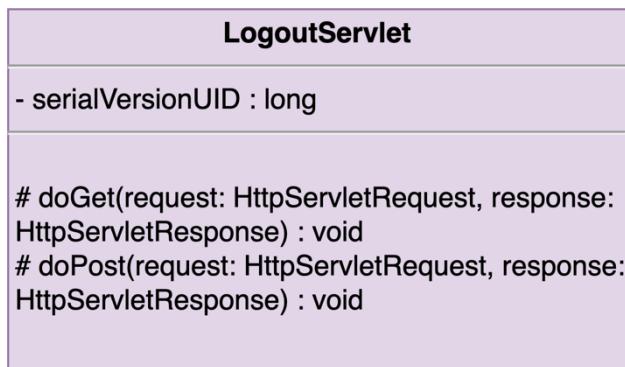


Figure 25: Class diagram of LogoutServlet

3.2.7. ModifyServiceServlet

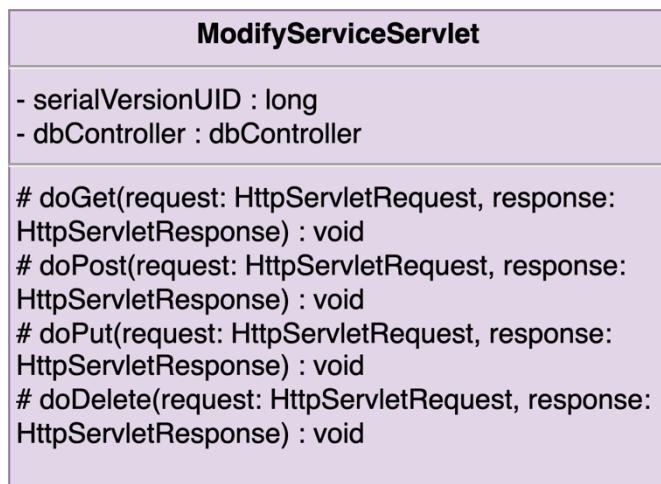


Figure 26: Class diagram of ModifyServiceServlet

3.2.8. ProfileServlet

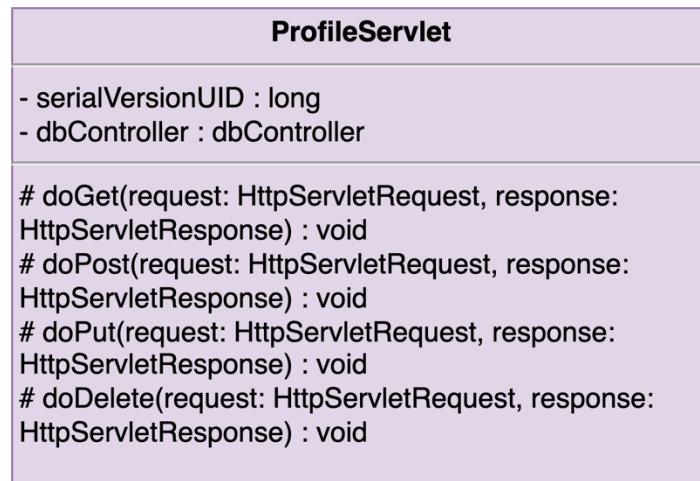


Figure 27: Class diagram of ProfileServlet

3.2.9. RegisterMentorServlet

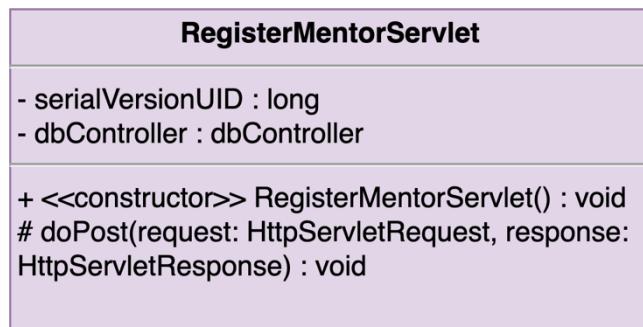


Figure 28: Class diagram of RegisterMentorServlet

3.2.10. RegisterSeekerServlet

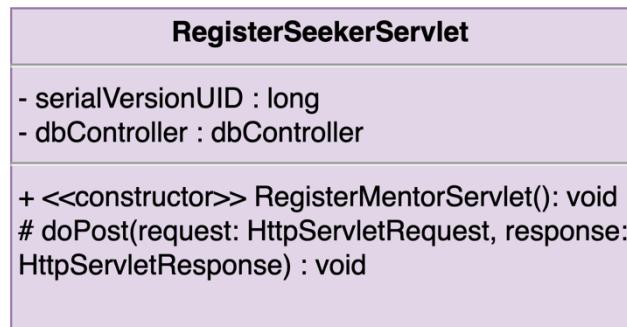


Figure 29: Class diagram of RegisterSeekerServlet

3.2.11. ServicesServlet

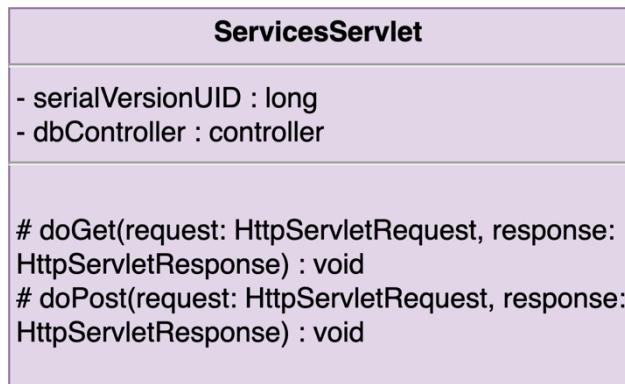


Figure 30: Class diagram of ServicesServlet

3.2.12. UpdatePasswordServlet



Figure 31: Class diagram of UpdatePasswordServlet

3.2.13. LoginModel

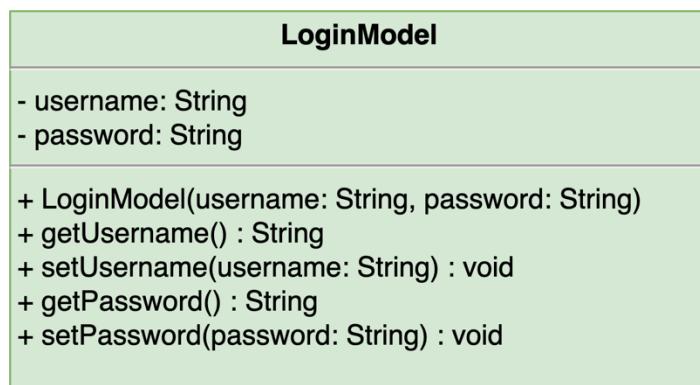


Figure 32: Class diagram of LoginModel

3.2.14. MentorModel

MentorModel
<pre> -name : String -username : String -email : String -phone : String -password : String -universityName : String -universityCountry : String -major : String -tuitionFee : int -scholarship : String -profilePhotoUrl : String </pre>
<pre> + <<constructor>> MentorModel() + <<constructor>> MentorModel(name: String, username: String, email: String, phone: String, password: String, universityName: String, universityCountry: String, major: String, tuitionFee: int, scholarship: String, imagePart: Part) + getName() : String + setName(name: String) : void + getUsername() : String + setUsername(username: String) : void + getEmail() : String + setEmail(email: String) : void + getPhone() : String + setPhone(phone: String) : void + getPassword() : String + setPassword(password: String) : void + getUniversityName() : String + setUniversityName(universityName: String) : void + getUniversityCountry() : String + setUniversityCountry(universityCountry: String) : void + getMajor() : String + setMajor(major: String) : void + getTuitionFee() : int + setTuitionFee(tuitionFee: int) : void + getScholarship() : String + setScholarship(scholarship: String) : void + getProfilePhotoUrl() : String + setProfilePhotoUrlFromPart(part: Part) : void + setProfilePhotoUrlFromDb(profilePhotoUrl: String) : void </pre>

Figure 33: Class diagram of MentorModel

3.2.15. SeekerModel

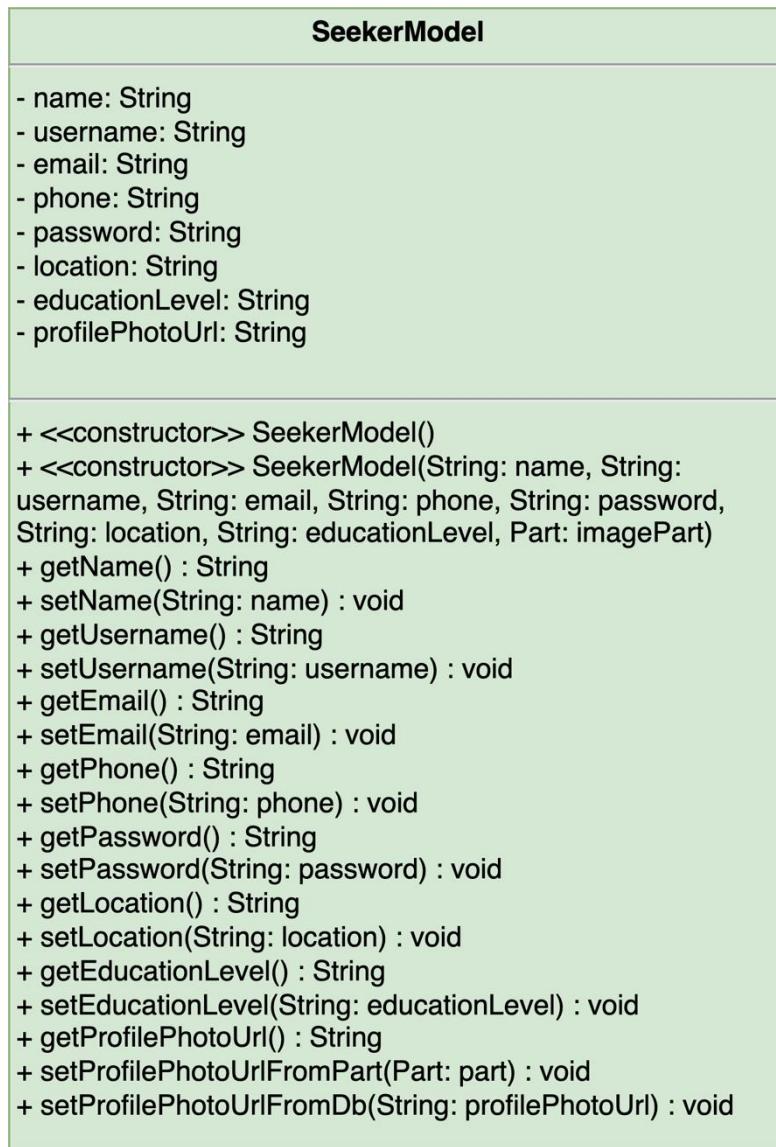


Figure 34: Class diagram of SeekerModel

3.2.16. ServiceModel

ServiceModel	
- id: int	
- title: String	
- description: String	
- price: int	
- creationDate: LocalDate	
- rating: float	
- users: int	
- status: String	
- mentorUsername: String	
+ <<constructor>> ServiceModel()	
+ <<constructor>> ServiceModel(int: id, String: title, String: description, int: price, LocalDate: creationDate, float: rating, int: users, String: status, String: mentorUsername)	
+ getId() : int	
+ setId(int: id) : void	
+ getTitle() : String	
+ setTitle(String: title) : void	
+ getDescription() : String	
+ setDescription(String: description) : void	
+ getPrice() : int	
+ setPrice(int: price) : void	
+ getCreationDate() : LocalDate	
+ setCreationDate(LocalDate: creationDate) : void	
+ getRating() : float	
+ setRating(float: rating) : void	
+ getUsers() : int	
+ setUsers(int: users) : void	
+ getStatus() : String	
+ setStatus(String: status) : void	
+ getMentorUsername() : String	
+ setMentorUsername(String: mentorUsername) : void	

Figure 35: Class diagram of ServiceModel

3.2.17. StringUtils

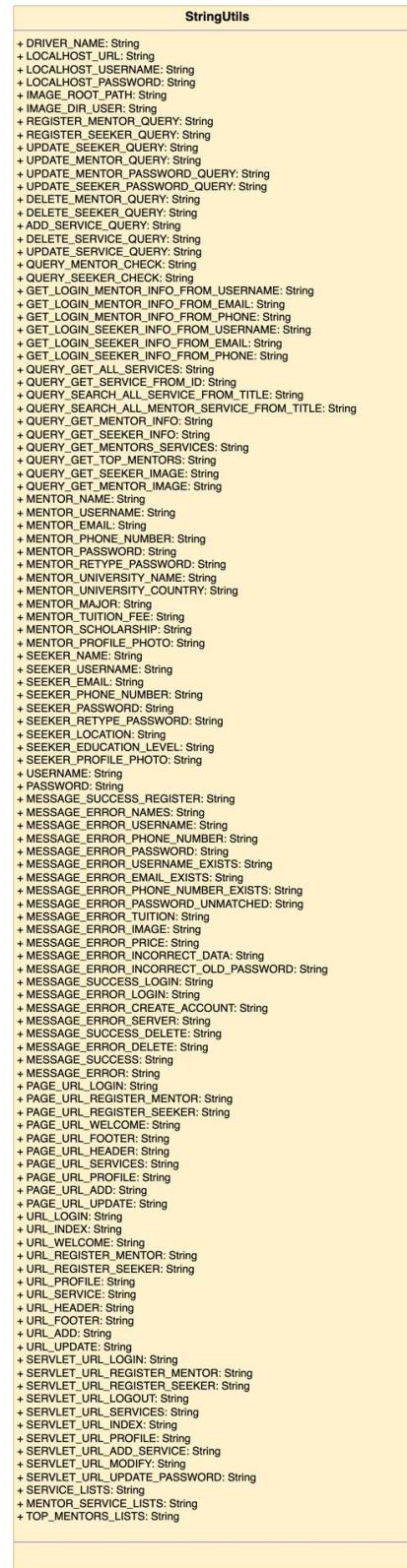


Figure 36: Class diagram of StringUtils

3.2.18. ValidationUtils

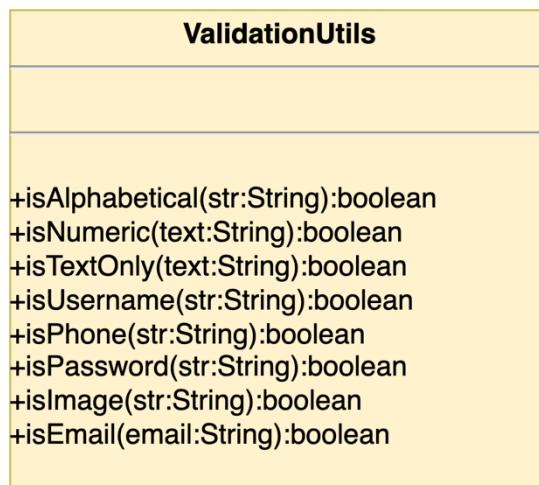


Figure 37: Class diagram of ValidationUtils

4. Method description

4.1. DbController

4.1.1. getConnection()

```
/*
 * Establishes a connection to the database.
 *
 * @return a Connection object representing the database connection.
 * @throws SQLException if a database access error occurs.
 * @throws ClassNotFoundException if the JDBC driver class is not found.
 */
public Connection getConnection() throws SQLException, ClassNotFoundException {

    Class.forName(StringUtils.DRIVER_NAME);
    return DriverManager.getConnection(StringUtils.LOCALHOST_URL, StringUtils.LOCALHOST_USERNAME,
        StringUtils.LOCALHOST_PASSWORD);
}
```

Figure 38: Screenshot of getConncetion() method

This method establishes a connection to a database using JDBC. It first loads the JDBC driver class specified in the StringUtils, and then creates a connection to the database using the URL, username, and password also specified in StringUtils. If an SQLException occurs during the connection process or if the JDBC driver class is not found, the method throws the corresponding exception.

4.1.2. registerMentor()

```

/*
 * Registers a mentor by adding their details to the database.
 *
 * @param mentor The MentorModel object containing the mentor's information.
 * @return 1 if the registration is successful, 0 if registration failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int registerMentor(MentorModel mentor) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.REGISTER_MENTOR_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, mentor.getName());
        stmt.setString(2, mentor.getUsername());
        stmt.setString(3, mentor.getEmail());
        stmt.setString(4, mentor.getPhone());
        stmt.setString(5, PasswordEncryptionWithAes.encrypt(mentor.getUsername(), mentor.getPassword()));
        stmt.setString(6, mentor.getUniversityName());
        stmt.setString(7, mentor.getUniversityCountry());
        stmt.setString(8, mentor.getMajor());
        stmt.setInt(9, mentor.getTuitionFee());
        stmt.setString(10, mentor.getScholarship());
        stmt.setString(11, mentor.getProfilePhotoUrl());

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 39: Screenshot of registerMentor() method

This method registers a mentor by inserting their details into the database using a prepared statement. It encrypts the mentor's password before storing it. The method returns 1 if the registration is successful, 0 if no rows are affected, -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.3. registerSeeker()

```

/**
 * Registers a seeker by adding their details to the database.
 *
 * @param seeker The SeekerModel object containing the seeker's information.
 * @return 1 if the registration is successful, 0 if registration failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int registerSeeker(SeekerModel seeker) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.REGISTER_SEEKER_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, seeker.getName());
        stmt.setString(2, seeker.getUsername());
        stmt.setString(3, seeker.getEmail());
        stmt.setString(4, seeker.getPhone());
        stmt.setString(5, PasswordEncryptionWithAes.encrypt(seeker.getUsername(), seeker.getPassword()));
        stmt.setString(6, seeker.getLocation());
        stmt.setString(7, seeker.getEducationLevel());
        stmt.setString(8, seeker.getProfilePhotoUrl());

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 40: Screenshot of registerSeeker() method

This method facilitates the registration of a seeker by inserting their details into the database using a prepared statement. It also encrypts the seeker's password before storing it. The method returns 1 if the registration is successful, 0 if no rows are affected, -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.4. getLoginInfo()

```


/**
 * Validates the username and password provided during login.
 *
 * @param loginModel The LoginModel object containing the username and password.
 * @return 1 if the login is successful for a mentor, 2 if successful for a seeker,
 *         0 if the username or password is incorrect, and -1 if the username is not found,
 *         -2 for internal errors.
 */
public int getLoginInfo(LoginModel loginModel) {

    // Try-catch block to handle potential SQL or ClassNotFoundException exceptions
    try {
        // Prepare a statement using the predefined query for mentor's login check
        PreparedStatement st_mentor = getConnection().prepareStatement(StringUtils.QUERY_MENTOR_CHECK);

        // Set the username in the first parameter of the prepared statement
        st_mentor.setString(1, loginModel.getUsername());

        // Execute the query and store the result set
        ResultSet result_m = st_mentor.executeQuery();

        // Check if there's a record returned from the query
        if (result_m.next()) {
            // Get the mentor's username from the database
            String userDb = result_m.getString(StringUtils.MENTOR_USERNAME);

            // Get the password from the database
            String passwordDb = result_m.getString(StringUtils.MENTOR_PASSWORD);

            // Decrypted password
            String decryptedPwd = PasswordEncryptionWithAes.decrypt(passwordDb, userDb);

            // Check if the username and password match the credentials from the database
            if (userDb.equals(loginModel.getUsername()) && decryptedPwd.equals(loginModel.getPassword())) {
                // Login successful, return 1
                return 1;
            } else {
                // Username or password mismatch, return 0
                return 0;
            }
        } else {
            // username not found in the mentor table, try seeker table

            // Prepare a statement using the predefined query for mentor's login check
            PreparedStatement st_seeker = getConnection().prepareStatement(StringUtils.QUERY_SEEKER_CHECK);

            // Set the username in the first parameter of the prepared statement
            st_seeker.setString(1, loginModel.getUsername());

            // Execute the query and store the result set
            ResultSet result = st_seeker.executeQuery();

            // Check if there's a record returned from the query
            if (result.next()) {
                // Get the mentor's username from the database
                String userDb = result.getString(StringUtils.SEEKER_USERNAME);

                // Get the password from the database
                String passwordDb = result.getString(StringUtils.SEEKER_PASSWORD);

                // Decrypted password
                String decryptedPwd = PasswordEncryptionWithAes.decrypt(passwordDb, userDb);

                // Check if the username and password match the credentials from the database
                if (userDb.equals(loginModel.getUsername()) && decryptedPwd.equals(loginModel.getPassword())) {
                    // Login successful, return 1
                    return 2;
                } else {
                    // Username or password mismatch, return 0
                    return 0;
                }
            } else {
                // username not found in the seeker table (both), return -1
                return -1;
            }
        }
    }

    // Catch SQLException and ClassNotFoundException if they occur
} catch (SQLException | ClassNotFoundException ex) {
    // Print the stack trace for debugging purposes
    ex.printStackTrace();
    // Return -2 to indicate an internal error
    return -2;
}
}


```

Figure 41: Screenshot of getAllLoginInfo() method

This method verifies login credentials for mentors and seekers by checking the provided username and password against stored data. It first attempts to authenticate mentors, then seekers if no mentor account matches. If authentication is successful for mentors, it returns 1; for seekers, it returns 2. If the provided credentials are incorrect or the username is not found, it returns 0 or -1, respectively. Internal errors result in a return of -2.

4.1.5. presentInMentor()

```
/*
 * Checks if the given value is present in the specified column of the mentor table.
 *
 * @param column The column in which to search for the value (username, email, or phone).
 * @param value The value to search for in the specified column.
 * @return true if the value is found in the specified column, false otherwise.
 */
public boolean presentInMentor(String column, String value) {
    try {
        PreparedStatement stmt;

        // preparing statement for getting info on the basis of column i.e user name,
        // email or phone
        switch (column) {
            case "mentor_username":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_MENTOR_INFO_FROM_USERNAME);
                break;
            case "mentor_email":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_MENTOR_INFO_FROM_EMAIL);
                break;
            case "mentor_phone":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_MENTOR_INFO_FROM_PHONE);
                break;
            default:
                stmt = null;
                break;
        }

        // Set the user name, email or phone number value in the prepared statement
        stmt.setString(1, value);

        // Execute the query and store the result set
        ResultSet result = stmt.executeQuery();

        // Return true if value found false otherwise
        return result.next();
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return false; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return false; // Server error
    }
}
```

Figure 42: Screenshot of presentInMentor() method

This method checks whether a given value is present in the specified column of the mentor table. It accepts parameters for the column to search in (username, email, or phone) and the corresponding value to search for. The method dynamically prepares a

SQL statement based on the specified column and executes it to query the database. If the value is found in the specified column, it returns true; otherwise, it returns false. It handles potential exceptions such as ClassNotFoundException or SQLException and returns false in case of internal or server errors.

4.1.6. presentInSeeker()

```
/*
 * Checks if the given value is present in the specified column of the seeker table.
 *
 * @param column The column in which to search for the value (username, email, or phone).
 * @param value The value to search for in the specified column.
 * @return true if the value is found in the specified column, false otherwise.
 */
public boolean presentInSeeker(String column, String value) {
    try {
        PreparedStatement stmt;

        // preparing statement for getting info on the basis of column i.e user name,
        // email or phone
        switch (column) {
            case "seeker_username":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_SEEKER_INFO_FROM_USERNAME);
                break;
            case "seeker_email":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_SEEKER_INFO_FROM_EMAIL);
                break;
            case "seeker_phone":
                stmt = getConnection().prepareStatement(StringUtils.GET_LOGIN_SEEKER_INFO_FROM_PHONE);
                break;
            default:
                stmt = null;
                break;
        }

        // Set the user name, email or phone number value in the prepared statement
        stmt.setString(1, value);

        // Execute the query and store the result set
        ResultSet result = stmt.executeQuery();

        // Return true if value found false otherwise
        return result.next();
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return false; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return false; // Server error
    }
}
```

Figure 43: Screenshot of presentInSeeker() method

This method checks whether a given value is present in the specified column of the seeker table. It accepts parameters for the column to search in (username, email, or phone) and the corresponding value to search for. The method dynamically prepares a SQL statement based on the specified column and executes it to query the database. If the value is found in the specified column, it returns true; otherwise, it returns false. It handles potential

exceptions such as ClassNotFoundException or SQLException and returns false in case of internal or server errors.

4.1.7. getAllServiceInfo()

```
/*
 * Retrieves information for all services present in the service table.
 *
 * @return An ArrayList containing ServiceModel objects representing the services.
 *         Returns null if an error occurs during database access.
 */
public ArrayList<ServiceModel> getAllServiceInfo(){
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_ALL_SERVICES);
        ResultSet result = stmt.executeQuery();

        ArrayList<ServiceModel> services = new ArrayList<ServiceModel>();

        while(result.next()) {
            ServiceModel service = new ServiceModel();
            service.setId(result.getInt("service_id"));
            service.setTitle(result.getString("service_title"));
            service.setDescription(result.getString("description"));
            service.setPrice(result.getInt("price"));
            service.setCreationDate(result.getDate("creation_date").toLocalDate());
            service.setRating(result.getFloat("rating"));
            service.setUsers(result.getInt("service_users"));
            service.setStatus(result.getString("status"));
            service.setMentorUsername(result.getString("mentor_username"));

            services.add(service);
        }
        return services;
    }catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}
```

Figure 44: Screenshot of getAllServiceInfo() method

This method retrieves information for all services stored in the service table of the database. It constructs an ArrayList of ServiceModel objects, creating each object with data retrieved from the database. Each ServiceModel object represents a service and contains attributes such as ID, title, description, price, creation date, rating, number of users, status, and mentor's username. If an error occurs during database access, such as SQL or class not found exceptions, the method returns null.

4.1.8. getAllMentorServiceInfo()

```


/**
 * Retrieves information for all services offered by a specific mentor.
 *
 * @param mentor_username The username of the mentor whose services are to be retrieved.
 * @return An ArrayList containing ServiceModel objects representing the mentor's services.
 *         Returns null if an error occurs during database access.
 */
public ArrayList<ServiceModel> getAllMentorServiceInfo(String mentor_username){
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_MENTORS_SERVICES);

        // Set the username in the prepared statement
        stmt.setString(1, mentor_username);

        ResultSet result = stmt.executeQuery();

        ArrayList<ServiceModel> mentor_services = new ArrayList<ServiceModel>();

        while(result.next()) {
            ServiceModel mentor_service = new ServiceModel();
            mentor_service.setId(result.getInt("service_id"));
            mentor_service.setTitle(result.getString("service_title"));
            mentor_service.setDescription(result.getString("description"));
            mentor_service.setPrice(result.getInt("price"));
            mentor_service.setCreationDate(result.getDate("creation_date").toLocalDate());
            mentor_service.setRating(result.getFloat("rating"));
            mentor_service.setUsers(result.getInt("service_users"));
            mentor_service.setStatus(result.getString("status"));
            mentor_service.setMentorUsername(result.getString("mentor_username"));

            mentor_services.add(mentor_service);
        }
        return mentor_services;
    }catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}


```

Figure 45: Screenshot of getAllMentorServiceInfo() method

This method retrieves information for all services offered by a specific mentor, identified by their username. It prepares a SQL statement to query the database for services associated with the given mentor username, sets the username parameter, and executes the query. The method then constructs an ArrayList of ServiceModel objects, each representing a service offered by the mentor. If any error occurs during database access, it returns null.

4.1.9. getMentorInfo()

```


/*
 * Retrieves mentor details from the database based on the passed username.
 *
 * @param mentor_username The username of the mentor whose details are to be retrieved.
 * @return A MentorModel object containing the mentor's details if found, or null if not found
 *         or if an error occurs during database access.
 */
public MentorModel getMentorInfo(String mentor_username){
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_MENTOR_INFO);

        // Set the username in the prepared statement
        stmt.setString(1, mentor_username);

        ResultSet result = stmt.executeQuery();

        if (result.next()) { // Move cursor to the first row
            MentorModel mentor_info = new MentorModel();
            mentor_info.setName(result.getString("mentor_name"));
            mentor_info.setUsername(result.getString("mentor_username"));
            mentor_info.setEmail(result.getString("mentor_email"));
            mentor_info.setPhone(result.getString("mentor_phone"));
            mentor_info.setPassword>PasswordEncryptionWithAes.decrypt(result.getString("mentor_password"), mentor_username));
            mentor_info.setUniversityName(result.getString("university_name"));
            mentor_info.setUniversityCountry(result.getString("university_country"));
            mentor_info.setMajor(result.getString("major"));
            mentor_info.setTuitionFee(result.getInt("tuition_fee"));
            mentor_info.setScholarship(result.getString("scholarship"));
            mentor_info.setProfilePhotoUrlFromDb(result.getString("mentor_photo"));

            return mentor_info;
        }
        return null;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}


```

Figure 46: Screenshot of getMentorInfo() method

This method retrieves mentor details from the database based on the provided username. It prepares a SQL statement, sets the username parameter, executes the query, and processes the ResultSet. If mentor details are found, a MentorModel object is created and populated with the retrieved information, including name, username, email, phone number, university details, tuition fee, scholarship, and profile photo URL. The mentor's password is decrypted using AES encryption before being set in the MentorModel object. If no details are found or an error occurs during database access, null is returned.

4.1.10. getSeekerInfo()

```

/**
 * Retrieves seeker details from the database based on the passed username.
 *
 * @param seeker_username The username of the seeker whose details are to be retrieved.
 * @return A SeekerModel object containing the seeker's details if found, or null if not found
 *         or if an error occurs during database access.
 */
public SeekerModel getSeekerInfo(String seeker_username){
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_SEEKER_INFO);

        // Set the username in the prepared statement
        stmt.setString(1, seeker_username);

        ResultSet result = stmt.executeQuery();

        if (result.next()) { // Move cursor to the first row
            SeekerModel seeker_info = new SeekerModel();
            seeker_info.setName(result.getString("seeker_name"));
            seeker_info.setUsername(result.getString("seeker_username"));
            seeker_info.setEmail(result.getString("seeker_email"));
            seeker_info.setPhone(result.getString("seeker_phone"));
            seeker_info.setPassword>PasswordEncryptionWithAes.decrypt(result.getString("seeker_password"), seeker_username);
            seeker_info.setLocation(result.getString("seeker_location"));
            seeker_info.setEducationLevel(result.getString("education_level"));
            seeker_info.setProfilePhotoUrlFromDb(result.getString("seeker_photo"));

            return seeker_info;
        }
        return null;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}

```

Figure 47: Screenshot of getSeekerInfo() method

This method retrieves seeker details from the database based on the provided username. It prepares a SQL statement, sets the username parameter, executes the query, and processes the ResultSet. If seeker details are found, a SeekerModel object is created and populated with the retrieved information, including name, username, email, phone number, location, and education level. The seeker's password is decrypted using AES encryption before being set in the SeekerModel object. If no details are found or an error occurs during database access, null is returned.

4.1.11. getSeekerImageName()

```

/*
 * Retrieves the image name (URL) of a seeker based on the passed username.
 *
 * @param username The username of the seeker whose image name is to be retrieved.
 * @return A string representing the image name (URL) of the seeker if found, or null if not found
 *         or if an error occurs during database access.
 */
public String getSeekerImageName(String username){

    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_SEEKER_IMAGE);

        // Set the username in the prepared statement
        stmt.setString(1, username);

        ResultSet result = stmt.executeQuery();

        if (result.next()) { // Move cursor to the first row
            return result.getString("seeker_photo");
        }
        return null;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}

```

Figure 48: Screenshot of getSeekerImageName() method

This method retrieves the image name (URL) of a seeker associated with the given username. It prepares a SQL statement, sets the username parameter, executes the query, and processes the ResultSet. If the seeker's image name is found, it returns the corresponding URL; otherwise, it returns null. Any encountered database access errors, such as SQLException or ClassNotFoundException, lead to a null return value.

4.1.12. getMentorImageName()

```
/**  
 * Retrieves the image name (URL) of a mentor based on the passed username.  
 *  
 * @param username The username of the mentor whose image name is to be retrieved.  
 * @return A string representing the image name (URL) of the mentor if found, or null if not found  
 *         or if an error occurs during database access.  
 */  
public String getMentorImageName(String username){  
  
    try {  
        PreparedStatement stmt = getConnection()  
            .prepareStatement(StringUtils.QUERY_GET_MENTOR_IMAGE);  
  
        // Set the username in the prepared statement  
        stmt.setString(1, username);  
  
        ResultSet result = stmt.executeQuery();  
  
        if (result.next()) { // Move cursor to the first row  
            return result.getString("mentor_photo");  
        }  
        return null;  
  
    }catch (SQLException | ClassNotFoundException ex) {  
        ex.printStackTrace();  
        return null;  
    }  
}
```

Figure 49: Screenshot of getMentorImageName() method

This method retrieves the image name (URL) of a mentor associated with the given username. It prepares a SQL statement, sets the username parameter, executes the query, and processes the ResultSet. If the mentor's image name is found, it returns the corresponding URL; otherwise, it returns null. Any encountered database access errors result in a null return value.

4.1.13. getTopMentorsInfo()

```


/*
 * Retrieves information for the top five mentors based on their average service rating.
 *
 * @return An ArrayList containing MentorModel objects representing the top five mentors.
 *         Returns null if an error occurs during database access.
 */
public ArrayList<MentorModel> getTopMentorsInfo() {
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_TOP_MENTORS);

        ResultSet result = stmt.executeQuery();

        ArrayList<MentorModel> topMentors = new ArrayList<MentorModel>();

        while(result.next()) {
            MentorModel mentor = new MentorModel();

            mentor.setName(result.getString("mentor_name"));
            mentor.setUsername(result.getString("mentor_username"));
            mentor.setEmail(result.getString("mentor_email"));
            mentor.setPhone(result.getString("mentor_phone"));
            mentor.setPassword(result.getString("mentor_password"));
            mentor.setUniversityName(result.getString("university_name"));
            mentor.setUniversityCountry(result.getString("university_country"));
            mentor.setMajor(result.getString("major"));
            mentor.setTuitionFee(result.getInt("tuition_fee"));
            mentor.setScholarship(result.getString("scholarship"));
            mentor.setProfilePhotoUrlFromDb(result.getString("mentor_photo"));

            topMentors.add(mentor);
        }
        return topMentors;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}


```

Figure 50: Screenshot of getTopMentorsInfo() method

This method fetches information for the top five mentors based on their average service rating. It prepares a SQL statement, executes the query, and constructs an ArrayList of MentorModel objects representing these mentors. Each MentorModel object contains details such as name, username, email, phone number, university information, tuition fee, scholarship, and profile photo URL. If any error occurs during database access, null is returned.

4.1.14. updateMentor()

```
/*
 * Updates a mentor's details in the database based on the old username and the updated MentorModel.
 *
 * @param username The old username of the mentor to be updated.
 * @param mentor The MentorModel object containing the updated mentor's information.
 * @return 1 if the update is successful, 0 if update failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int updateMentor(String username, MentorModel mentor) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.UPDATE_MENTOR_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, mentor.getName());
        stmt.setString(2, mentor.getUsername());
        stmt.setString(3, mentor.getEmail());
        stmt.setString(4, mentor.getPhone());
        stmt.setString(5, PasswordEncryptionWithAes.encrypt(mentor.getUsername(), mentor.getPassword()));
        stmt.setString(6, mentor.getProfilePhotoUrl());

        stmt.setString(7, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}
```

Figure 51: Screenshot of updateMentor() method

This method updates a mentor's details in the database based on the old username and the updated MentorModel. It prepares a SQL statement using a predefined query for mentor updates. After setting the updated mentor's information in the prepared statement, including name, username, email, phone number, encrypted password, and profile photo URL, it executes the update statement. The method returns 1 if the update is successful, 0 if the update failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.15. updateSeeker()

```

/**
 * Updates a seeker's details in the database based on the old username and the updated SeekerModel.
 *
 * @param username The old username of the seeker to be updated.
 * @param seeker The SeekerModel object containing the updated seeker's information.
 * @return 1 if the update is successful, 0 if update failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int updateSeeker(String username, SeekerModel seeker) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.UPDATE_SEEKER_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, seeker.getName());
        stmt.setString(2, seeker.getUsername());
        stmt.setString(3, seeker.getEmail());
        stmt.setString(4, seeker.getPhone());
        stmt.setString(5, PasswordEncryptionWithAes.encrypt(seeker.getUsername(), seeker.getPassword()));
        stmt.setString(6, seeker.getProfilePhotoUrl());

        stmt.setString(7, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 52: Screenshot of updateSeekerMethod()

This method updates a seeker's details in the database based on the old username and the updated SeekerModel. It prepares a SQL statement using a predefined query for seeker updates. After setting the updated seeker's information in the prepared statement, including name, username, email, phone number, encrypted password, and profile photo URL, it executes the update statement. The method returns 1 if the update is successful, 0 if the update failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.16. deleteMentor()

```

/**
 * Deletes a mentor from the mentor table based on the passed username.
 *
 * @param username The username of the mentor to be deleted.
 * @return 1 if the deletion is successful, 0 if deletion failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int deleteMentor(String username) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.DELETE_MENTOR_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 53: Screenshot of deleteMentor() method

This method deletes a mentor from the mentor table based on the passed username. It prepares a SQL statement using a predefined query for mentor deletion. After setting the username parameter in the prepared statement, it executes the delete statement. The method returns 1 if the deletion is successful, 0 if the deletion failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.17. deleteSeeker()

```

/*
 * Deletes a seeker from the seeker table based on the passed username.
 *
 * @param username The username of the seeker to be deleted.
 * @return 1 if the deletion is successful, 0 if deletion failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int deleteSeeker(String username) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.DELETE_SEEKER_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }

    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 54: Screenshot of deleteSeeker() method

This method deletes a seeker from the seeker table based on the passed username. It prepares a SQL statement using a predefined query for seeker deletion. After setting the username parameter in the prepared statement, it executes the delete statement. The method returns 1 if the deletion is successful, 0 if the deletion failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.18. addService()

```


    /**
     * Adds a service to the database in the service table based on the passed ServiceModel and mentor username.
     *
     * @param service      The ServiceModel object representing the service to be added.
     * @param mentor_username The username of the mentor offering the service.
     * @return 1 if the addition is successful, 0 if addition failed, -1 for internal errors,
     *         and -2 for server errors.
     */
    public int addService(ServiceModel service, String mentor_username) {
        try {
            // Prepare a statement using the predefined query for mentor registration
            PreparedStatement stmt = getConnection().prepareStatement(StringUtils.ADD_SERVICE_QUERY);

            // Set the student information in the prepared statement
            stmt.setString(1, service.getTitle());
            stmt.setString(2, service.getDescription());
            stmt.setInt(3, service.getPrice());

            LocalDate currDate = LocalDate.now();
            stmt.setDate(4, Date.valueOf(currDate));

            stmt.setString(5, mentor_username);

            // Execute the update statement and store the number of affected rows
            int result = stmt.executeUpdate();

            // Check if the update was successful (i.e., at least one row affected)
            if (result > 0) {
                return 1; // Registration successful
            } else {
                return 0; // Registration failed (no rows affected)
            }
        } catch (ClassNotFoundException | SQLException ex) {
            // Print the stack trace for debugging purposes
            ex.printStackTrace();
            return -1; // Internal error
        } catch (Exception ex) {
            ex.printStackTrace();
            return -2; // Server error
        }
    }
}


```

Figure 55: Screenshot of addService() method

This method adds a service to the database in the service table based on the passed ServiceModel and mentor username. It prepares a SQL statement using a predefined query for adding a service. After setting the service information in the prepared statement, including title, description, price, and creation date, it executes the update statement. The method returns 1 if the addition is successful, 0 if the addition failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.19. deleteService()

```

/**
 * Deletes a service from the database in the service table based on the passed service ID.
 *
 * @param service_id The ID of the service to be deleted.
 * @return 1 if the deletion is successful, 0 if deletion failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int deleteService(int service_id) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.DELETE_SERVICE_QUERY);

        // Set the student information in the prepared statement
        stmt.setInt(1, service_id);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 56: Screenshot of deleteService() method

This method deletes a service from the database in the service table based on the passed service ID. It prepares a SQL statement using a predefined query for deleting a service. After setting the service ID parameter in the prepared statement, it executes the delete statement. The method returns 1 if the deletion is successful, 0 if the deletion failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.20. updateService()

```

/**
 * Updates a service in the database based on the passed ServiceModel and service ID.
 *
 * @param service The updated ServiceModel object representing the service.
 * @param service_id The ID of the service to be updated.
 * @return 1 if the update is successful, 0 if update failed, -1 for internal errors,
 *         and -2 for server errors.
 */
public int updateService(ServiceModel service, int service_id) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.UPDATE_SERVICE_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, service.getTitle());
        stmt.setString(2, service.getDescription());
        stmt.setInt(3, service.getPrice());

        stmt.setInt(4, service_id);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }

    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}

```

Figure 57: Screenshot of updateService() method

This method updates a service in the database based on the passed ServiceModel and service ID. It prepares a SQL statement using a predefined query for updating a service. After setting the updated service information, including title, description, and price, and the service ID in the prepared statement, it executes the update statement. The method returns 1 if the update is successful, 0 if the update failed (no rows affected), -1 for internal errors such as ClassNotFoundException or SQLException, and -2 for server errors caused by other exceptions.

4.1.21. getServiceInfoFromId()

```

/**
 * Retrieves service details from the database based on the given service ID.
 *
 * @param service_id The ID of the service to retrieve.
 * @return A ServiceModel object representing the service details if found, otherwise null.
 */
public ServiceModel getServiceInfoFromId(int service_id){
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_GET_SERVICE_FROM_ID);

        // Set the username in the prepared statement
        stmt.setInt(1, service_id);

        ResultSet result = stmt.executeQuery();

        if (result.next()) { // Move cursor to the first row
            ServiceModel service = new ServiceModel();
            service.setId(result.getInt("service_id"));
            service.setTitle(result.getString("service_title"));
            service.setDescription(result.getString("description"));
            service.setPrice(result.getInt("price"));
            service.setCreationDate(result.getDate("creation_date").toLocalDate());
            service.setRating(result.getFloat("rating"));
            service.setUsers(result.getInt("service_users"));
            service.setStatus(result.getString("status"));
            service.setMentorUsername(result.getString("mentor_username"));

            return service;
        }
        return null;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}

```

Figure 58: Screenshot of `getServiceInfoFromId()` method

This method retrieves service details from the database based on the given service ID. It prepares a SQL statement using a predefined query to fetch service information using the service ID. After setting the service ID in the prepared statement, it executes the query and retrieves the result set. If a record is found, it creates a ServiceModel object and sets its attributes with the retrieved data. Finally, it returns the ServiceModel object representing the service details if found, otherwise returns null.

4.1.22. getSearchedServiceInfo()

```

/**
 * Searches for services in the database based on the provided service title.
 *
 * @param service_title The title of the service to search for.
 * @return An ArrayList of ServiceModel objects representing the found services, or null if an error occurs.
 */
public ArrayList<ServiceModel> getSearchedServiceInfo(String service_title) {
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_SEARCH_ALL_SERVICE_FROM_TITLE);

        // Set the username in the prepared statement
        stmt.setString(1, service_title + "%");

        ResultSet result = stmt.executeQuery();

        ArrayList<ServiceModel> searched_services = new ArrayList<ServiceModel>();

        while(result.next()) {
            ServiceModel found_service = new ServiceModel();
            found_service.setId(result.getInt("service_id"));
            found_service.setTitle(result.getString("service_title"));
            found_service.setDescription(result.getString("description"));
            found_service.setPrice(result.getInt("price"));
            found_service.setCreationDate(result.getDate("creation_date").toLocalDate());
            found_service.setRating(result.getFloat("rating"));
            found_service.setUsers(result.getInt("service_users"));
            found_service.setStatus(result.getString("status"));
            found_service.setMentorUsername(result.getString("mentor_username"));

            searched_services.add(found_service);
        }
        return searched_services;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}

```

Figure 59: Screenshot of getSearchedServiceInfo() method

This method searches services in the database based on a given service title. It constructs a search query using the title provided, allowing for partial matches. After executing the query, it collects the results and creates ServiceModel objects for each found service, filling in their details. Finally, it returns a list of these ServiceModel objects representing the matched services, or null if there's an issue accessing the database.

4.1.23. getSearchedServiceInfoMentor()

```

/*
 * Searches for services offered by a specific mentor in the database based on the provided service title and mentor username.
 *
 * @param service_title The title of the service to search for.
 * @param mentor_username The username of the mentor offering the service.
 * @return An ArrayList of ServiceModel objects representing the found services, or null if an error occurs.
 */
public ArrayList<ServiceModel> getSearchedServiceInfoMentor(String service_title, String mentor_username) {
    try {
        PreparedStatement stmt = getConnection()
            .prepareStatement(StringUtils.QUERY_SEARCH_ALL_MENTOR_SERVICE_FROM_TITLE);

        // Set the username in the prepared statement
        stmt.setString(1, mentor_username);
        stmt.setString(2, service_title + "%");

        ResultSet result = stmt.executeQuery();

        ArrayList<ServiceModel> searched_services = new ArrayList<ServiceModel>();

        while(result.next()) {
            ServiceModel found_service = new ServiceModel();
            found_service.setId(result.getInt("service_id"));
            found_service.setTitle(result.getString("service_title"));
            found_service.setDescription(result.getString("description"));
            found_service.setPrice(result.getInt("price"));
            found_service.setCreationDate(result.getDate("creation_date").toLocalDate());
            found_service.setRating(result.getFloat("rating"));
            found_service.setUsers(result.getInt("service_users"));
            found_service.setStatus(result.getString("status"));
            found_service.setMentorUsername(result.getString("mentor_username"));

            searched_services.add(found_service);
        }
        return searched_services;
    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
        return null;
    }
}

```

Figure 60: Screenshot of getSearchedServiceInfoMentor() method

This method searches for services offered by a specific mentor in the database based on the provided service title and mentor username. It constructs a search query using both the service title and mentor username, allowing for partial matches in the title. After executing the query, it collects the results and creates ServiceModel objects for each found service, populating their details. Finally, it returns a list of these ServiceModel objects representing the matched services, or null if there's an issue accessing the database.

4.1.24. updateMentorPassword()

```


/**
 * Updates the password of a mentor in the database.
 *
 * @param username      The username of the mentor whose password is to be updated.
 * @param new_password  The new password to set for the mentor.
 * @return 1 if the password update is successful, 0 if update failed, -1 for internal errors, and -2 for server errors.
 */
public int updateMentorPassword(String username, String new_password) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.UPDATE_MENTOR_PASSWORD_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, PasswordEncryptionWithAes.encrypt(username, new_password));
        stmt.setString(2, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}


```

Figure 61: Screenshot of updateMentorPassword() method

This method updates the password of a mentor in the database. It takes the mentor's username and the new password as input parameters. Upon execution, it encrypts the new password using AES encryption, updates the password in the database for the specified mentor, and returns 1 if the password update is successful. If the update fails, it returns 0. Internal errors are indicated by returning -1, and server errors are indicated by returning -2.

4.1.25. updateSeekerPassword()

```
/**
 * Updates the password of a seeker in the database.
 *
 * @param username The username of the seeker whose password is to be updated.
 * @param new_password The new password to set for the seeker.
 * @return 1 if the password update is successful, 0 if update failed, -1 for internal errors, and -2 for server errors.
 */
public int updateSeekerPassword(String username, String new_password) {
    try {
        // Prepare a statement using the predefined query for mentor registration
        PreparedStatement stmt = getConnection().prepareStatement(StringUtils.UPDATE_SEEKER_PASSWORD_QUERY);

        // Set the student information in the prepared statement
        stmt.setString(1, PasswordEncryptionWithAes.encrypt(username, new_password));
        stmt.setString(2, username);

        // Execute the update statement and store the number of affected rows
        int result = stmt.executeUpdate();

        // Check if the update was successful (i.e., at least one row affected)
        if (result > 0) {
            return 1; // Registration successful
        } else {
            return 0; // Registration failed (no rows affected)
        }
    } catch (ClassNotFoundException | SQLException ex) {
        // Print the stack trace for debugging purposes
        ex.printStackTrace();
        return -1; // Internal error
    } catch (Exception ex) {
        ex.printStackTrace();
        return -2; // Server error
    }
}
```

Figure 62: Screenshot of updateSeekerPassword() method

This method is responsible for updating the password of a seeker in the database. It accepts the seeker's username and the new password as parameters. Upon execution, it encrypts the new password using AES encryption, updates the password in the database for the specified seeker, and returns 1 if the password update is successful. If the update fails, it returns 0. Internal errors are indicated by returning -1, and server errors are indicated by returning -2.

4.2. AuthenticationFilter

4.2.1. init()

```
/**
 * Initializes the filter.
 * @param filterConfig The filter configuration.
 * @throws ServletException If the initialization fails.
 */
@Override
public void init(FilterConfig filterConfig) throws ServletException {
```

Figure 63: Screenshot of init() method

This method initializes the filter with the provided configuration. It sets up any necessary resources or parameters needed for the filter to function. If initialization fails, it throws a ServletException.

4.2.2. doFilter()

```
/*
 * Filters incoming requests and performs authentication and authorization checks.
 * @param request The HttpServletRequest object.
 * @param response The HttpServletResponse object.
 * @param chain The FilterChain object for invoking the next filter in the chain.
 * @throws IOException If an I/O error occurs.
 * @throws ServletException If the request processing fails.
 */
@Override
public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
    throws IOException, ServletException {
    // Cast request and response objects to HttpServletRequest and HttpServletResponse for type safety
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;

    // Get the requested URI
    String uri = req.getRequestURI();

    // Allow access to static resources (CSS) and the index page without checking login
    if (uri.endsWith(".css") || uri.endsWith(".png") || uri.endsWith(".jpg")) {
        chain.doFilter(request, response);
        return;
    }

    if (uri.endsWith("/")) {
        request.getRequestDispatcher(StringUtil.SERVLET_URL_INDEX).forward(request, response);
        return;
    }

    // Separate flags for login, login/logout servlets, and register page/servlet for better readability
    boolean isLoggedInPage = uri.endsWith(StringUtil.PAGE_URL_LOGIN);
    boolean isLoginServlet = uri.endsWith(StringUtil.SERVLET_URL_LOGIN);
    boolean isLogoutServlet = uri.endsWith(StringUtil.SERVLET_URL_LOGOUT);

    boolean isRegisterPage = (uri.endsWith(StringUtil.PAGE_URL_REGISTER_MENTOR) || uri.endsWith(StringUtil.PAGE_URL_REGISTER_SEEKER));
    boolean isRegisterServlet = (uri.endsWith(StringUtil.SERVLET_URL_REGISTER_MENTOR) || uri.endsWith(StringUtil.SERVLET_URL_REGISTER_SEEKER));

    // Separate flag for index page and servlet for better readability
    boolean isIndexPage = uri.endsWith(StringUtil.URL_INDEX);
    boolean isIndexServlet = uri.endsWith(StringUtil.SERVLET_URL_INDEX);

    // Separate flag for other pages for better readability
    boolean isProfilePage = uri.endsWith(StringUtil.URL_PROFILE);
    boolean isServicePage = uri.endsWith(StringUtil.URL_SERVICE);
    boolean isHeaderPage = uri.endsWith(StringUtil.URL_HEADER);
    boolean isFooterPage = uri.endsWith(StringUtil.URL_FOOTER);

    // Separate flag for other pages for better readability
    boolean isLoginPage = uri.endsWith(StringUtil.URL_PROFILE);
    boolean isServicePage = uri.endsWith(StringUtil.URL_SERVICE);
    boolean isHeaderPage = uri.endsWith(StringUtil.URL_HEADER);
    boolean isFooterPage = uri.endsWith(StringUtil.URL_FOOTER);

    // Check if a session exists and if the username attribute is set to determine login status
    HttpSession session = req.getSession(false); // Don't create a new session if one doesn't exist
    boolean isLoggedInIn = session != null && session.getAttribute(StringUtil.USERNAME) != null;

    // FOR FOR NEW/LOGGED OUT USERS
    if (!isLoggedInIn) {
        // Redirect to login if trying to access a protected resource
        if (!isLoggedInPage || isLoginServlet || isRegisterPage || isIndexServlet || isIndexPage) {
            res.sendRedirect(req.getContextPath() + StringUtil.PAGE_URL_LOGIN);
        }

        // Redirect to index servlet if trying to access index page
        else if (isIndexPage) {
            request.getRequestDispatcher(StringUtil.SERVLET_URL_INDEX).forward(request, response);
        }

        // Allow access to the requested resource
        else {
            chain.doFilter(request, response);
        }
    }

    // FOR LOGGED IN USERS
    else if (isLoggedInIn) {
        // Redirect to the welcome page if trying to access login page again
        if (isLoggedInPage || isRegisterPage || isHeaderPage || isFooterPage) {
            res.sendRedirect(req.getContextPath() + StringUtil.PAGE_URL_WELCOME);
        }

        // Redirect to index servlet if trying to access index page
        else if (isIndexPage) {
            request.getRequestDispatcher(StringUtil.SERVLET_URL_INDEX).forward(request, response);
        }

        // Redirect to services servlet if trying to access service page
        else if (isServicePage) {
            request.getRequestDispatcher(StringUtil.SERVLET_URL_SERVICES).forward(request, response);
        }

        // Redirect to services servlet if trying to access service page
        else if (isProfilePage) {
            request.getRequestDispatcher(StringUtil.SERVLET_URL_PROFILE).forward(request, response);
        }
    }
}
```

```

        // Allow access to the requested resource
    } else {
        chain.doFilter(request, response);
    }

} else {
    // Allow access to the requested resource if user is logged in or accessing unprotected resources
    chain.doFilter(request, response);
}

}

```

Figure 64: Screenshot of doFilter() method

The doFilter method intercepts incoming requests, executes authentication and authorization checks, and then either allows the request to proceed or redirects the user as needed. It passes the request down the filter chain for further processing. This method may throw IOException or ServletException for errors.

4.2.3. destroy()

```

/**
 * Cleans up resources used by the filter.
 */
@Override
public void destroy() {
}

```

Figure 65: Screenshot of destroy() method

The destroy method cleans up resources used by the filter. It's called when the filter is being taken out of service, releasing any held resources to ensure a graceful shutdown.

4.3. AddServiceServlet

4.3.1. AddServiceServlet()

```

/**
 * Initializes the AddServiceServlet by creating a new instance of DbController.
 * @see HttpServlet#HttpServlet()
 */
public AddServiceServlet() {
    this.dbController = new DbController();
}

```

Figure 66: Screenshot of AddServiceServlet() constructor

The AddServiceServlet begins by setting up a new object of DbController to deal with the database.

4.3.2. doPost()

```

/**
 * Handles HTTP POST requests sent to the servlet, adding a new service to the database.
 * Retrieves attributes from the current session, extracts service information from request parameters,
 * creates a ServiceModel object for the new service, and adds the service to the database using DbController.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");

    // Extract service information from add service page request parameters
    String title = request.getParameter("title");
    String description = request.getParameter("description");
    int price = Integer.parseInt(request.getParameter("price"));

    // creating a service model for new added service
    ServiceModel newService = new ServiceModel();
    newService.setTitle(title);
    newService.setDescription(description);
    newService.setPrice(price);

    // Adding the register mentor(model) in database
    int result = dbController.addService(newService, currentUser);

    //checking for exception
    if (result > 0) {
        response.sendRedirect(request.getContextPath() + StringUtils.SERVLET_URL_SERVICES);
    } else {
        // Code will be written in later weeks (ERROR 404 PAGE)
    }
}

```

Figure 67: Screenshot of doPost() method

This method handles HTTP POST requests by adding a new service to the database. It first retrieves attributes from the current session, such as the username of the current user. Then, it extracts service information from the request parameters, including the title, description, and price of the service. Next, it creates a ServiceModel object to represent the new service. This object is then passed to the DbController to add the service to the database. If the addition is successful, the method redirects to the services page. Otherwise, error handling will be implemented in future updates.

4.4. IndexServlet

4.4.1. doGet()

```


/**
 * Handles HTTP GET requests.
 * Retrieves top mentors' information from the database, adds it to the request attribute,
 * and forwards the request to the index page for display.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Get the top mentors info from database and store in topMentors arraylist
    ArrayList<MentorModel> topMentors = controller.getTopMentorsInfo();

    // adding the arraylist in request attribute to display in index page
    request.setAttribute(StringUtils.TOP_MENTORS_LISTS, topMentors);

    // Redirect to index page with attributes in request
    request.getRequestDispatcher(StringUtils.URL_INDEX).forward(request, response);
}


```

Figure 68: Screenshot of doGet() method

This method handles HTTP GET requests. It retrieves information about the top mentors from the database and stores it in an ArrayList named topMentors. This ArrayList is then added to the request attribute so that it can be displayed on the index page. Finally, the method forwards the request to the index page for display.

4.5. LoginServlet

4.5.1. LoginServlet()

```


/**
 * Initializes the LoginServlet by creating a new instance of DbController,
 * enabling interaction with the underlying database.
 *
 * @see HttpServlet#HttpServlet()
 */
public LoginServlet() {
    this.dbController = new DbController();
}


```

Figure 69: Screenshot of LoginServlet() constructor

This constructor initializes the LoginServlet with a new DbController instance, enabling database interaction for handling login requests and validating user credentials.

4.5.2. doPost()

```


    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     * Handles HTTP POST requests.
     * Extracts username and password from the request parameters, validates login credentials,
     * sets session attributes for logged-in users, and redirects accordingly.
     * @param request The HttpServletRequest object representing the request.
     * @param response The HttpServletResponse object representing the response.
     * @throws ServletException If the servlet encounters a ServletException.
     * @throws IOException If an I/O error occurs while processing the request.
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        // Extract username and password from the request parameters
        String userName = request.getParameter(StringUtils.USERNAME);
        String password = request.getParameter(StringUtils.PASSWORD);

        // Create a LoginModel object to hold user credentials
        LoginModel loginModel = new LoginModel(userName, password);

        // Call DBController to validate login credentials
        int loginResult = dbController.getLoginInfo(loginModel);

        // Handle login results with appropriate messages and redirects

        if (loginResult == 1) { // FOR MENTOR (LOGIN SUCCESS)

            // getting image from database
            String mentorImage = dbController.getMentorImageName(userName);

            // creating session and setting attributes
            HttpSession userSession = request.getSession();
            userSession.setAttribute("type", "mentor");
            userSession.setAttribute("username", userName);
            userSession.setAttribute("image", mentorImage);
            userSession.setMaxInactiveInterval(30*30);

            // creating cookie
            Cookie userCookie= new Cookie("user", userName);
            userCookie.setMaxAge(30*60);
            response.addCookie(userCookie);

            // redirect to welcome page
            response.sendRedirect(request.getContextPath() + StringUtils.PAGE_URL_WELCOME);

        } else if (loginResult == 2) { // FOR SEEKER (LOGIN SUCCESS)

            // getting image from database
            String seekerImage = dbController.getSeekerImageName(userName);

            // creating session and setting attributes
            HttpSession userSession = request.getSession();
            userSession.setAttribute("type", "seeker");
            userSession.setAttribute("username", userName);
            userSession.setAttribute("image", seekerImage);
            userSession.setMaxInactiveInterval(30*30);

            // creating cookie
            Cookie userCookie= new Cookie("user", userName);
            userCookie.setMaxAge(30*60);
            response.addCookie(userCookie);

            // redirect to welcome page
            response.sendRedirect(request.getContextPath() + StringUtils.PAGE_URL_WELCOME);

        } else if (loginResult == 0) {
            // Username or password mismatch
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_LOGIN);
            request.getRequestDispatcher(StringUtils.PAGE_URL_LOGIN).forward(request, response);
        }

        } else if (loginResult == -1) {
            // Username not found
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_CREATE_ACCOUNT);
            request.getRequestDispatcher(StringUtils.PAGE_URL_LOGIN).forward(request, response);
        }

        } else {
            // Internal server error
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
            request.getRequestDispatcher(StringUtils.PAGE_URL_LOGIN).forward(request, response);
        }
    }
}


```

Figure 70: Screenshot of doPost() method

This method manages HTTP POST requests. It begins by extracting the username and password from the request parameters. Then, it creates a LoginModel object to hold these credentials and calls on the DBController to validate them. Depending on the result, it proceeds with appropriate actions: for successful logins (either mentor or seeker), it sets session attributes and redirects to the welcome page. If the username or password doesn't match, it displays an error message on the login page. If the username is not found, it prompts the user to create an account. Any other unexpected errors are handled with a generic server error message.

4.6. LogoutServlet

4.6.1. doGet()

```
/*
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP GET requests by redirecting to the doPost method.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Redirect to doPost method
    doPost(request, response);
}
```

Figure 71: Screenshot of doGet() method

This method handles HTTP GET requests by redirecting them to the doPost method. When a GET request is received, it simply calls the doPost method to process the request.

4.6.2. doPost()

```


/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP POST requests.
 * Clears existing cookies, invalidates user session (if it exists), and redirects to the index page.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Handle logout request

    // Clearing existing cookies
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            // Set max age to 0 to effectively delete the cookie
            cookie.setMaxAge(0);
            response.addCookie(cookie);
        }
    }

    // Invalidate user session (if it exists)
    HttpSession session = request.getSession(false);
    if (session != null) {
        session.invalidate();
    }

    // Redirect to index page
    response.sendRedirect(request.getContextPath() + StringUtils.SERVLET_URL_INDEX);
}


```

Figure 72: Screenshot of doPost() method

This method handles HTTP POST requests for logging out users. It clears existing cookies and invalidates the user session, if active, and then redirects to the index page. This ensures a secure logout process and returns users to the application's main page.

4.7. ModifyServiceServlet

4.7.1. doGet()

```


/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP GET requests by retrieving service information and redirecting to the update page.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Get selected service from request, get all its info from database and store in service as service model
    ServiceModel service = dbController.getServiceInfoFromId(Integer.parseInt(request.getParameter("update_service_id")));

    // adding the service in request attribute to display in update page
    request.setAttribute("updateService", service);

    // Redirect to update page with attributes in request
    request.getRequestDispatcher(StringUtils.PAGE_URL_UPDATE).forward(request, response);
}


```

Figure 73: Screenshot of doGet() method

This method handles GET requests by retrieving details of a specific service for updating. It extracts the service's ID from the request, fetches its details from the database, and

stores them in a ServiceModel object. Then, it forwards this object to the update page for display. Users can conveniently modify service details through this page.

4.7.2. doPost()

```
/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP POST requests by processing update or delete actions.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // getting id's from hidden input parameters
    String updateId = request.getParameter("update_service_id");
    String deleteId = request.getParameter("delete_service_id");

    // if update button clicked
    if (updateId != null && !updateId.isEmpty()) {
        doPut(request, response);
    }

    // if delete button clicked
    if (deleteId != null && !deleteId.isEmpty()) {
        doDelete(request, response);
    }
}
```

Figure 74 : Screenshot of doPost() method

This method manages HTTP POST requests, specifically handling update or delete actions for services. Upon receiving a POST request, it first retrieves the IDs of services to be updated or deleted from hidden input parameters. If the update button is clicked and an ID is provided, the method invokes the doPut method to process the update action. Similarly, if the delete button is clicked and an ID is provided, the method invokes the doDelete method to handle the delete action.

4.7.3. doPut()

```

/**
 * @see HttpServlet#doPut(HttpServletRequest, HttpServletResponse)
 * Handles HTTP PUT requests by updating service information in the database.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Extract user information from update page request parameters
    String title = request.getParameter("title");
    String description = request.getParameter("description");
    int price = Integer.parseInt(request.getParameter("price"));

    // Create a service model for updated service
    ServiceModel updatedService = new ServiceModel();
    updatedService.setTitle(title);
    updatedService.setDescription(description);
    updatedService.setPrice(price);

    // Updating the service in database
    int result = dbController.updateService(updatedService, Integer.parseInt(request.getParameter("update_service_id")));

    //Checking for exception when updating
    if (result > 0) {
        response.sendRedirect(request.getContextPath() + StringUtils.SERVLET_URL_SERVICES);
    } else {
        // Internal server error
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
    }
}

```

Figure 75: Screenshot of doPut() method

This method handles HTTP PUT requests by updating service information in the database. It extracts details of the service to be updated from the request parameters, including the title, description, and price. These details are used to create a ServiceModel object representing the updated service. The method then updates the service in the database using the DbController. If the update is successful, the method redirects to the services page. Otherwise, it sets an internal server error message to be displayed.

4.7.4. doDelete()

```


/**
 * @see HttpServlet#doDelete(HttpServletRequest, HttpServletResponse)
 * Handles HTTP DELETE requests by deleting services from the database.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Deleting the selected service from database
    if (dbController.deleteService(Integer.parseInt(request.getParameter("delete_service_id")))) == 1) {

        // send success message through request and redirect to services servlet
        request.setAttribute(StringUtil.MESSAGE_SUCCESS, StringUtil.MESSAGE_SUCCESS_DELETE);
        response.sendRedirect(request.getContextPath() + StringUtil.SERVLET_URL_SERVICES);

        // If deletion failed
    } else {
        // Internal server error
        request.setAttribute(StringUtil.MESSAGE_ERROR, StringUtil.MESSAGE_ERROR_SERVER);
    }
}


```

Figure 76: Screenshot of `doDelete()` method

This method handles HTTP DELETE requests by deleting services from the database. It retrieves the ID of the service to be deleted from the request parameters and proceeds to delete the corresponding service from the database using the DbController. If the deletion is successful, it sets a success message to be displayed and redirects to the services servlet, allowing users to view the updated list of services. If the deletion fails, it sets an internal server error message to be displayed.

4.8. ProfileServlet

4.8.1. doGet()

```

/*
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP GET requests by retrieving user profile information and forwarding to the profile page.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");

    // FOR MENTOR (get current mentor info from database as mentor model and set attribute in request)
    if (currentUserType.equals("mentor")) {
        MentorModel mentor = dbController.getMentorInfo(currentUser);
        request.setAttribute("userProfile", mentor);
    }

    // FOR SEEKER (get current seeker info from database as seeker model and set attribute in request)
    else {
        SeekerModel seeker = dbController.getSeekerInfo(currentUser);
        request.setAttribute("userProfile", seeker);
    }

    // Redirect to profile page with attributes in request
    request.getRequestDispatcher(StringUtils.PAGE_URL_PROFILE).forward(request, response);
}

```

Figure 77: Screenshot of doGet() method

This method handles GET requests by fetching user profile information based on the current session. It distinguishes between mentor and seeker types, retrieves the respective profile data from the database, and sets it as a request attribute. Finally, it forwards the request to the profile page for display.

4.8.2. doPost()

```

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 * Handles HTTP POST requests for updating user profile or deleting user account.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Checking if update password button clicked
    Object isUpdate = request.getAttribute("isUpdate");
    if (isUpdate != null) {
        if((Integer)isUpdate == 1) {
            doGet(request, response);
        }
    }

    // If update details or delete button clicked
} else {

    // getting username from input parameters
    String updateUsername = request.getParameter("update_username");
    String deleteUsername = request.getParameter("delete_username");

    // if update button clicked
    if (updateUsername != null && !updateUsername.isEmpty()) {
        doPut(request, response);
    }

    // if delete button clicked
    if (deleteUsername != null && !deleteUsername.isEmpty()) {
        doDelete(request, response);
    }
}
}

```

Figure 78: Screenshot of doPost() method

This method handles HTTP POST requests for updating user profiles or deleting user accounts. It checks if the update password button is clicked; if yes, it redirects to the doGet method. Otherwise, it processes update details or delete button clicks based on the provided username, invoking the doPut or doDelete method accordingly.

4.8.3. doPut()

```

/**
 * @see HttpServlet#doPut(HttpServletRequest, HttpServletResponse)
 * Handles HTTP PUT requests for updating user profile information.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doPut(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");
    String currentImage = (String) userSession.getAttribute("image");

    // Extract user information from profile page request parameters
    String name = request.getParameter("name");
    String username = request.getParameter("update_username");
    String email = request.getParameter("email");
    String phone = request.getParameter("phone");
    String password = request.getParameter("password");
    Part photo = request.getPart("photo");

    // validation flag
    boolean is_valid = true;

    // full name validation
    if (!ValidationUtils.isAlphabetical(name)) {
        is_valid = false;
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_NAMES);
    }

    // user name validation
    if (!ValidationUtils.isUsername(username)) {
        is_valid = false;
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_USERNAME);
    }

    // phone validation
    if (!ValidationUtils.isPhone(phone)) {
        is_valid = false;
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_PHONE_NUMBER);
    }

    // image extension validation
    if (!ValidationUtils.isImage(photo.getSubmittedFileName()) && !(photo.getSubmittedFileName() == "")) {
        is_valid = false;
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_IMAGE);
    }
}

```

```

// if not valid then send to do get method and return
if (!isValid) {
    doGet(request, response);
    return;
}

// FOR MENTOR
if (currentUserType.equals("mentor")) {

    // Create a mentor model for updated mentor
    MentorModel updatedMentor = new MentorModel();
    updatedMentor.setName(name);
    updatedMentor.setUsername(username);
    updatedMentor.setEmail(email);
    updatedMentor.setPhone(phone);
    updatedMentor.setPassword(password);

    // checking if photo updated
    if (photo.getSize() == 0) {
        updatedMentor.setProfilePhotoUrlFromDb(currentImage);
    } else {
        updatedMentor.setProfilePhotoUrlFromPart(photo);
    }

    // Updating the registered mentor(model) in database
    int result = dbController.updateMentor(currentUser, updatedMentor);

    // checking for exception when updating
    if (result > 0) {

        // Upload new image in tomcat server
        String savePath = StringUtils.IMAGE_DIR_USER;
        String fileName = updatedMentor.getProfilePhotoUrl();

        if (!(photo.getSize() == 0))
            photo.write(savePath + fileName);

        // updating username and image in session
        userSession.setAttribute("username", username);
        userSession.setAttribute("image", fileName);

        // redirect to welcome page
        request.getRequestDispatcher(StringUtils.PAGE_URL_WELCOME).forward(request, response);
    } else {
        // Internal server error
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
    }
}

// FOR SEEKER
else {

```

```

// Create a seeker model for updated seeker
SeekerModel updatedSeeker = new SeekerModel();
updatedSeeker.setName(name);
updatedSeeker.setUsername(username);
updatedSeeker.setEmail(email);
updatedSeeker.setPhone(phone);
updatedSeeker.setPassword(password);

// checking if photo updated
if (photo.getSize() == 0) {
    updatedSeeker.setProfilePhotoUrlFromDb(currentImage);
} else {
    updatedSeeker.setProfilePhotoUrlFromPart(photo);
}

// Updating the registered seeker(model) in database
int result = dbController.updateSeeker(currentUser, updatedSeeker);

// checking for exception when updating
if (result > 0) {

    // Upload new image in tomcat server
    String savePath = StringUtils.IMAGE_DIR_USER;
    String fileName = updatedSeeker.getProfilePhotoUrl();
    if (!(photo.getSize() == 0))
        photo.write(savePath + fileName);

    // updating username and image in session
    userSession.setAttribute("username", username);
    userSession.setAttribute("image", fileName);

    // redirect to welcome page
    request.getRequestDispatcher(StringUtils.PAGE_URL_WELCOME).forward(request, response);

} else {
    // Internal server error
    request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
}
}

```

Figure 79: Screenshot of doPut() method

This method handles HTTP PUT requests for updating user profile information. It retrieves user attributes from the session and user input from request parameters. Validation checks are performed, and if data is invalid, it redirects to the doGet method. For mentors and seekers, respective model objects are created with updated information. If a new photo is uploaded, it's saved; otherwise, the current photo URL is retained. Updated user information is then sent to the database for updating. If successful, the session is updated, and users are redirected to the welcome page. Otherwise, an internal server error message is displayed.

4.8.4. doDelete()

```


/**
 * @see HttpServlet#doDelete(HttpServletRequest, HttpServletResponse)
 * Handles HTTP DELETE requests for deleting user account.
 * @param request The HttpServletRequest object representing the request.
 * @param response The HttpServletResponse object representing the response.
 * @throws ServletException If the servlet encounters a ServletException.
 * @throws IOException If an I/O error occurs while processing the request.
 */
protected void doDelete(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");

    // FOR MENTOR
    if (currentUserType.equals("mentor")) {

        // deleting the registered mentor(model) from database
        int result = dbController.deleteMentor(currentUser);

        // checking for exception when deleting
        if (result > 0) {
            request.getRequestDispatcher(StringUtils.SERVLET_URL_LOGOUT).forward(request, response);
        } else {
            // Internal server error
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
        }
    }

    // FOR SEEKER
    else {

        // deleting the registered seeker(model) from database
        int result = dbController.deleteSeeker(currentUser);

        // checking for exception when deleting
        if (result > 0) {
            request.getRequestDispatcher(StringUtils.SERVLET_URL_LOGOUT).forward(request, response);
        } else {
            // Internal server error
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
        }
    }
}


```

Figure 80: Screenshot of doDelete() method

This method handles HTTP DELETE requests for deleting user accounts. It retrieves user attributes from the current session, including the username and user type. Depending on whether the user is a mentor or seeker, it proceeds to delete the corresponding user account from the database using the DbController. Upon successful deletion, the method redirects to the logout servlet to log the user out. If any exceptions occur during the deletion process, an internal server error message is displayed.

4.9. RegisterMentorServlet

4.9.1. RegisterMentorServlet()

```
/*
 * Constructs a new RegisterMentorServlet.
 * Initializes a new instance of DbController for database operations.
 * @see HttpServlet#HttpServlet()
 */
public RegisterMentorServlet() {
    this.dbController = new DbController();
}
```

Figure 81: Screenshot of RegisterMentorServlet() constructor

This method initializes a new instance of the RegisterMentorServlet class and creates a new DbController object for handling database operations.

4.9.2. doPost()

```
/*
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 * Handles POST requests for registering mentors.
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException      if an I/O error occurs
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Extract mentor information from register page request parameters
    String name = request.getParameter(StringUtils.MENTOR_NAME);
    String username = request.getParameter(StringUtils.MENTOR_USERNAME);
    String email = request.getParameter(StringUtils.MENTOR_EMAIL);
    String phone = request.getParameter(StringUtils.MENTOR_PHONE_NUMBER);
    String password = request.getParameter(StringUtils.MENTOR_PASSWORD);
    String retypePassword = request.getParameter(StringUtils.MENTOR_RETYPE_PASSWORD);
    String universityName = request.getParameter(StringUtils.MENTOR_UNIVERSITY_NAME);
    String universityCountry = request.getParameter(StringUtils.MENTOR_UNIVERSITY_COUNTRY);
    String major = request.getParameter(StringUtils.MENTOR_MAJOR);
    int tuitionFee = Integer.parseInt(request.getParameter(StringUtils.MENTOR_TUITION_FEE));
    String scholarship = request.getParameter(StringUtils.MENTOR_SCHOLARSHIP);
    Part profilePhoto = request.getPart(StringUtils.MENTOR_PROFILE_PHOTO);

    // validation flag
    boolean is_valid = true;

    // message to store all error messages
    StringBuilder errorMessage = new StringBuilder();

    // full name validation
    if (!ValidationUtils.isAlphabetical(name)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_NAMES);
    }

    // user name validation
    if (!ValidationUtils.isUsername(username)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_USERNAME);
    }

    // phone validation
    if (!ValidationUtils.isPhone(phone)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_PHONE_NUMBER);
    }
}
```

```

// password validation
if (!ValidationUtils.isPassword(password)) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PASSWORD);
}
if (!password.equals(retypePassword)) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PASSWORD_UNMATCHED);
}

// tuition fee validation
if (!ValidationUtils.isNumeric(Integer.toString(tuitionFee))) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_TUITION);
}

// image extension validation
if (!ValidationUtils.isImage(profilePhoto.getSubmittedFileName() && !(profilePhoto.getSubmittedFileName() == "")) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_IMAGE);
}

// checking if user name exists
boolean username_exist = dbController.presentInMentor("mentor_username", username);
if (username_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_USERNAME_EXISTS);
}

// checking if email exists
boolean email_exist = dbController.presentInMentor("mentor_email", email);
if (email_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_EMAIL_EXISTS);
}

// checking if number exists
boolean number_exist = dbController.presentInMentor("mentor_phone", phone);
if (number_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PHONE_NUMBER_EXISTS);
}

// if input is invalid then
if (!is_valid) {
    // set error request attribute
    request.setAttribute(StringUtils.MESSAGE_ERROR, errorMessage.toString());
}

// if every input is valid then
else {

    // Create model for registered mentor
    MentorModel mentor = new MentorModel(name, username, email, phone, password, universityName,
        universityCountry, major, tuitionFee, scholarship, profilePhoto);

    // Adding the register mentor(model) in database
    int result = dbController.registerMentor(mentor);

    //checking for exception
    if (result > 0) {

        // Upload image in tomcat server
        String savePath = StringUtils.IMAGE_DIR_USER;
        String fileName = mentor.getProfilePhotoUrl();
        if (!(fileName.equals("default-profile.png")))
            profilePhoto.write(savePath + fileName);

        // set success request attribute
        request.setAttribute(StringUtils.MESSAGE_SUCCESS, StringUtils.MESSAGE_SUCCESS_REGISTER);
    } else {
        // Internal server error
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
    }
}

// Redirect to register mentor page with request attributes
request.getRequestDispatcher(StringUtils.PAGE_URL_REGISTER_MENTOR).forward(request, response);
}

```

Figure 82: Screenshot of doPost() method

This method handles POST requests for registering mentors. It extracts mentor information from the request parameters, performs validation checks, and processes the registration accordingly. If the input is valid, it creates a MentorModel object and adds it to the database. Upon successful registration, it uploads the profile photo to the server and sets a success message attribute. If any errors occur during the process, it sets an error message attribute. Finally, it redirects to the register mentor page with the appropriate request attributes.

4.10. RegisterSeekerServlet

4.10.1. RegisterSeekerServlet()

```
/**  
 * Constructs a new RegisterSeekerServlet.  
 * Initializes a new instance of DbController for database operations.  
 * @see HttpServlet#HttpServlet()  
 */  
public RegisterSeekerServlet() {  
    this.dbController = new DbController();  
}
```

Figure 83: Screenshot of RegisterSeekerServlet() constructor

This method initializes a new instance of the RegisterSeekerServlet class and creates a new DbController object for handling database operations.

4.10.2. doPost()

```
/*
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 * Handles POST requests for registering seekers.
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException      if an I/O error occurs
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Extract mentor information from register page request parameters
    String name = request.getParameter(StringUtils.SEEKER_NAME);
    String username = request.getParameter(StringUtils.SEEKER_USERNAME);
    String email = request.getParameter(StringUtils.SEEKER_EMAIL);
    String phone = request.getParameter(StringUtils.SEEKER_PHONE_NUMBER);
    String password = request.getParameter(StringUtils.SEEKER_PASSWORD);
    String retypePassword = request.getParameter(StringUtils.SEEKER_RETYPE_PASSWORD);
    String location = request.getParameter(StringUtils.SEEKER_LOCATION);
    String educationLevel = request.getParameter(StringUtils.SEEKER_EDUCATION_LEVEL);
    Part profilePhoto = request.getPart(StringUtils.SEEKER_PROFILE_PHOTO);

    // validation flag
    boolean is_valid = true;

    // message to store all error messages
    StringBuilder errorMessage = new StringBuilder();

    // full name validation
    if (!ValidationUtils.isAlphabetical(name)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_NAMES);
    }

    // user name validation
    if (!ValidationUtils.isUsername(username)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_USERNAME);
    }

    // phone validation
    if (!ValidationUtils.isPhone(phone)) {
        is_valid = false;
        errorMessage.append(StringUtils.MESSAGE_ERROR_PHONE_NUMBER);
    }
}
```

```

// password validation
if (!Validationutils.isPassword(password)) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PASSWORD);
}
if (!password.equals(retypePassword)) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PASSWORD_UNMATCHED);
}

// image extension validation
if (!Validationutils.isImage(profilePhoto.getSubmittedFileName()) && !(profilePhoto.getSubmittedFileName() == ""))
{
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_IMAGE);
}

// checking if user name exists
boolean username_exist = dbController.presentInSeeker("seeker_username", username);
if (username_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_USERNAME_EXISTS);
}

// checking if email exists
boolean email_exist = dbController.presentInSeeker("seeker_email", email);
if (email_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_EMAIL_EXISTS);
}

// checking if number exists
boolean number_exist = dbController.presentInSeeker("seeker_phone", phone);
if (number_exist) {
    is_valid = false;
    errorMessage.append(StringUtils.MESSAGE_ERROR_PHONE_NUMBER_EXISTS);
}

// if input is invalid then
if (!is_valid) {
    // set error request attribute
    request.setAttribute(StringUtils.MESSAGE_ERROR, errorMessage.toString());
}

// if every input is valid then
else {

    // Create model for registered seeker
    SeekerModel seeker = new SeekerModel(name, username, email, phone, password, location, educationLevel, profilePhoto);

    // Adding the register seeker(model) in database
    int result = dbController.registerSeeker(seeker);

    //checking for exception
    if (result == 1) {

        // Upload image in tomcat server
        String savePath = StringUtils.IMAGE_DIR_USER;
        String fileName = seeker.getProfilePhotoUrl();
        if(!(fileName.equals("default-profile.png")))
            profilePhoto.write(savePath + fileName);

        // set success request attribute
        request.setAttribute(StringUtils.MESSAGE_SUCCESS, StringUtils.MESSAGE_SUCCESS_REGISTER);
    } else {
        // Internal server error
        request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
    }
}

// Redirect to register seeker page with request attributes
request.getRequestDispatcher(StringUtils.PAGE_URL_REGISTER_SEEKER).forward(request, response);
}

```

Figure 84: Screenshot of doPost() method

This method handles POST requests for registering seekers. It extracts seeker information from the request parameters, performs validation checks, and processes the registration accordingly. If the input is valid, it creates a SeekerModel object and adds it to the database. Upon successful registration, it uploads the profile photo to the server

and sets a success message attribute. If any errors occur during the process, it sets an error message attribute. Finally, it redirects to the register seeker page with the appropriate request attributes.

4.11. ServicesServlet

4.11.1. doGet()

```
/** Handles GET requests to display services based on user type (seeker or mentor)
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");

    // Extract search information from register page request parameters
    String searchKey = request.getParameter("search");

    // FOR SEEKER
    if (currentUserType.equals("seeker")) {

        // Declaring services ArrayList to store all services
        ArrayList<ServiceModel> services;

        // Get searched services if search button clicked
        if (searchKey == null) {
            services = controller.getAllServiceInfo();
        }

        // Get all services for default
        else {
            services = controller.getSearchServiceInfo(searchKey);
        }

        // adding ArrayList of services in request attribute to display in services.jsp
        request.setAttribute(StringUtils.SERVICE_LISTS, services);
        request.setAttribute("isSeeker", true);

        // Initializing and declaring hashMap to connect mentor details with their service
        HashMap<String, MentorModel> hashMap = new HashMap<>();

        // Loop on each items on ArrayList
        for (ServiceModel service: services) {
            // Add mentor name as key and its model as value
            String currUsername = service.getMentorUsername();
            MentorModel mentor = controller.getMentorInfo(currUsername);
            hashMap.put(service.getMentorUsername(), mentor);
        }

        // adding the mentor hashmap in request attribute to display in services page
        request.setAttribute("mentorMap", hashMap);
    }
}
```

```

// FOR MENTOR
else {

    // Declaring services arraylist to store mentor's all services
    ArrayList<ServiceModel> mentor_services;

    // Get searched services if search button clicked
    if (searchKey == null) {
        mentor_services = controller.getAllMentorServiceInfo(currentUser);
    }

    // Get all mentor's services for default
    else {
        mentor_services = controller.getSearchServiceInfoMentor(searchKey, currentUser);
    }

    // adding arraylist of mentor's services in request attribute to display in services.jsp
    request.setAttribute(StringUtils.MENTOR_SERVICE_LISTS, mentor_services);
    request.setAttribute("isMentor", true);

    // Initializing and declaring hashMap to connect mentor details with their service
    HashMap<String, MentorModel> hashMap = new HashMap<>();

    // Loop on each items on arraylist
    for (ServiceModel service: mentor_services) {
        // Add mentor name as key and its model as value
        String currUsername = service.getMentorUsername();
        MentorModel mentor = controller.getMentorInfo(currUsername);
        hashMap.put(service.getMentorUsername(), mentor);
    }

    // adding the mentor hashmap in request attribute to display in services page
    request.setAttribute("mentorMap", hashMap);
}

// Redirect to services page with attributes in request
request.getRequestDispatcher(StringUtils.PAGE_URL_SERVICES).forward(request, response);
}

```

Figure 85: Screenshot of doGet() method

This method handles GET requests to display services based on the user type, either seeker or mentor. It retrieves attributes from the current session to determine the user type. Then, it extracts search information from the request parameters. For seekers, it retrieves and displays all services or searched services, along with associated mentor details. For mentors, it retrieves and displays the mentor's services or searched services, also with associated mentor details. Finally, it redirects to the services page with the necessary attributes in the request.

4.11.2. doPost()

```

/**
 * Handles POST requests by delegating to doGet method
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}

```

Figure 86: Screenshot of doPost() method

This method handles POST requests by simply delegating to the doGet method.

4.12. UpdatePasswordServlet

4.12.1. doPost()

```
/*
 * Handles POST requests to update passwords
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException      if an I/O error occurs
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");

    // getting old and new password from request parameters
    String oldPassword = request.getParameter("old_password");
    String newPassword = request.getParameter("new_password");

    // Getting the correct(previous) password from the database
    String correctPassword;
    if (currentUserType.equals("mentor")) { // FOR MENTOR
        correctPassword = dbController.getMentorInfo(currentUser).getPassword();
    } else { // FOR SEEKER
        correctPassword = dbController.getSeekerInfo(currentUser).getPassword();
    }

    // If old password entered matches password in database
    if (oldPassword.equals(correctPassword)) {

        // validation for new password
        if(!ValidationUtils.isPassword(newPassword)) {
            request.setAttribute(StringUtil.MESSAGE_ERROR, StringUtil.MESSAGE_ERROR_PASSWORD);
            request.setAttribute("isUpdate", 1);
            request.getRequestDispatcher(StringUtil.SERVLET_URL_PROFILE).include(request, response);
            return;
        }

        // Put method if successful validation
        doPut(request, response);
    }

    // redirect to profile servlet with error message if old password does not match
    else {
        request.setAttribute(StringUtil.MESSAGE_ERROR, StringUtil.MESSAGE_ERROR_INCORRECT_OLD_PASSWORD);
        request.setAttribute("isUpdate", 1);
        request.getRequestDispatcher(StringUtil.SERVLET_URL_PROFILE).include(request, response);
    }
}
```

Figure 87: Screenshot of doPost() method

This method handles POST requests for updating passwords. It verifies the old password provided by the user against the one stored in the database. If the old password matches, it validates the new password and delegates to the doPut method for updating the password. If the old password does not match, it redirects to the profile servlet with an error message.

4.12.2. doPut()

```


/**
 * Handles PUT requests to update passwords in the database
 *
 * @param request The HttpServletRequest object
 * @param response The HttpServletResponse object
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 * @see HttpServlet#doPut(HttpServletRequest, HttpServletResponse)
 */
protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");

    // FOR MENTOR
    if (currentUserType.equals("mentor")) {

        // Update the mentor in database
        int result = dbController.updateMentorPassword(currentUser, request.getParameter("new_password"));

        //checking for exception when updating
        if (result > 0) {
            request.setAttribute("isUpdate", 1);
            request.getRequestDispatcher(StringUtils.PAGE_URL_WELCOME).include(request, response);
        } else {
            // Internal server error
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
        }
    }

    // FOR SEEKER
    else {

        // Update the seeker in database
        int result = dbController.updateSeekerPassword(currentUser, request.getParameter("new_password"));

        //checking for exception when updating
        if (result > 0) {
            request.setAttribute("isUpdate", 1);
            request.getRequestDispatcher(StringUtils.PAGE_URL_WELCOME).include(request, response);
        } else {
            // Internal server error
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_SERVER);
        }
    }
}


```

Figure 88: Screenshot of doPut() method

This method handles PUT requests to update passwords in the database. It retrieves the current user's information from the session, determines the user type (mentor or seeker), and updates the password accordingly in the database. If the password update is successful, it redirects to the welcome page. Otherwise, it sets an internal server error message.

4.13. LoginModel

4.13.1. LoginModel

```
/**  
 * Constructs a new LoginModel object with the given username and password.  
 *  
 * @param username The username of the user  
 * @param password The password of the user  
 */  
public LoginModel(String username, String password) {  
    this.username = username;  
    this.password = password;  
}
```

Figure 89: Screenshot of LoginModel() constructor

This constructor initializes a new instance of the LoginModel class with the provided username and password.

4.13.2. Getter and Setter methods

```
/**  
 * Returns the username of the user.  
 *  
 * @return The username of the user  
 */  
public String getUsername() {  
    return username;  
}  
  
/**  
 * Sets the username of the user.  
 *  
 * @param username The new username to set  
 */  
public void setUsername(String username) {  
    this.username = username;  
}  
  
/**  
 * Returns the password of the user.  
 *  
 * @return The password of the user  
 */  
public String getPassword() {  
    return password;  
}  
  
/**  
 * Sets the password of the user.  
 *  
 * @param password The new password to set  
 */  
public void setPassword(String password) {  
    this.password = password;  
}
```

Figure 90: Screenshot of getter and setter methods

The getUsername() method retrieves the username of the user, while setUsername(String username) sets a new username. Similarly, getPassword() retrieves the user's password, and setPassword(String password) updates it. These methods facilitate access to and modification of user credentials, ensuring smooth authentication and account management within the application.

4.14. MentorModel

4.14.1. MentorModel

```

/*
 * Default constructor for MentorModel.
 */
public MentorModel() {
}

/**
 * Constructs a MentorModel object with the specified attributes.
 *
 * @param name          The name of the mentor
 * @param username      The username of the mentor
 * @param email         The email of the mentor
 * @param phone         The phone number of the mentor
 * @param password      The password of the mentor
 * @param universityName The name of the mentor's university
 * @param universityCountry The country of the mentor's university
 * @param major          The major of the mentor
 * @param tuitionFee    The tuition fee of the mentor
 * @param scholarship    The scholarship information of the mentor
 * @param imagePart      The image part for the mentor's profile photo
 */
public MentorModel(String name, String username, String email, String phone, String password, String universityName,
                   String universityCountry, String major, int tuitionFee, String scholarship, Part imagePart) {
    super();
    this.name = name;
    this.username = username;
    this.email = email;
    this.phone = phone;
    this.password = password;
    this.universityName = universityName;
    this.universityCountry = universityCountry;
    this.major = major;
    this.tuitionFee = tuitionFee;
    this.scholarship = scholarship;
    this.profilePhotoUrl = getImageUrl(imagePart);
}

```

Figure 91: Screenshot of MentorModel() constructor

The MentorModel class comprises two constructors. The default constructor initializes a new instance of MentorModel without requiring any parameters. The second constructor is more detailed, accepting parameters for attributes such as the mentor's name, username, email, phone number, password, university details, tuition fee, scholarship information, and profile photo. This constructor allows for the creation of MentorModel objects with specific attribute values, facilitating ease of use and flexibility in the initialization process

4.14.2. Getter and Setter methods

```
/**  
 * Returns the name of the mentor.  
 *  
 * @return The name of the mentor  
 */  
public String getName() {  
    return name;  
}  
  
/**  
 * Sets the name of the mentor.  
 *  
 * @param name The name of the mentor to set  
 */  
public void setName(String name) {  
    this.name = name;  
}  
  
/**  
 * Returns the username of the mentor.  
 *  
 * @return The username of the mentor  
 */  
public String getUsername() {  
    return username;  
}  
  
/**  
 * Sets the username of the mentor.  
 *  
 * @param username The username of the mentor to set  
 */  
public void setUsername(String username) {  
    this.username = username;  
}  
  
/**  
 * Returns the email of the mentor.  
 *  
 * @return The email of the mentor  
 */  
public String getEmail() {  
    return email;  
}
```

```
/**  
 * Sets the email of the mentor.  
 *  
 * @param email The email of the mentor to set  
 */  
public void setEmail(String email) {  
    this.email = email;  
}  
  
/**  
 * Returns the phone number of the mentor.  
 *  
 * @return The phone number of the mentor  
 */  
public String getPhone() {  
    return phone;  
}  
  
/**  
 * Sets the phone number of the mentor.  
 *  
 * @param phone The phone number of the mentor to set  
 */  
public void setPhone(String phone) {  
    this.phone = phone;  
}  
  
/**  
 * Returns the password of the mentor.  
 *  
 * @return The password of the mentor  
 */  
public String getPassword() {  
    return password;  
}  
  
/**  
 * Sets the password of the mentor.  
 *  
 * @param password The password of the mentor to set  
 */  
public void setPassword(String password) {  
    this.password = password;  
}  
  
/**  
 * Returns the name of the mentor's university.  
 *  
 * @return The name of the mentor's university  
 */  
public String getUniversityName() {  
    return universityName;  
}
```

```
/**  
 * Sets the name of the mentor's university.  
 *  
 * @param universityName The name of the mentor's university to set  
 */  
public void setUniversityName(String universityName) {  
    this.universityName = universityName;  
}  
  
/**  
 * Returns the country of the mentor's university.  
 *  
 * @return The country of the mentor's university  
 */  
public String getUniversityCountry() {  
    return universityCountry;  
}  
  
/**  
 * Sets the country of the mentor's university.  
 *  
 * @param universityCountry The country of the mentor's university to set  
 */  
public void setUniversityCountry(String universityCountry) {  
    this.universityCountry = universityCountry;  
}  
  
/**  
 * Returns the major of the mentor.  
 *  
 * @return The major of the mentor  
 */  
public String getMajor() {  
    return major;  
}  
  
/**  
 * Sets the major of the mentor.  
 *  
 * @param major The major of the mentor to set  
 */  
public void setMajor(String major) {  
    this.major = major;  
}  
  
/**  
 * Returns the tuition fee of the mentor.  
 *  
 * @return The tuition fee of the mentor  
 */  
public int getTuitionFee() {  
    return tuitionFee;  
}
```

```

    /**
     * Sets the tuition fee of the mentor.
     *
     * @param tuitionFee The tuition fee of the mentor to set
     */
    public void setTuitionFee(int tuitionFee) {
        this.tuitionFee = tuitionFee;
    }

    /**
     * Returns the scholarship information of the mentor.
     *
     * @return The scholarship information of the mentor
     */
    public String getScholarship() {
        return scholarship;
    }

    /**
     * Sets the scholarship information of the mentor.
     *
     * @param scholarship The scholarship information of the mentor to set
     */
    public void setScholarship(String scholarship) {
        this.scholarship = scholarship;
    }

    /**
     * Returns the profile photo url of the mentor.
     *
     * @return The profile photo url of the mentor
     */
    public String getProfilePhotoUrl() {
        return profilePhotoUrl;
    }

    /**
     * Sets the profile photo URL of the mentor using a Part object.
     *
     * @param part The Part object representing the mentor's profile photo
     */
    public void setProfilePhotoUrlFromPart(Part part) {
        this.profilePhotoUrl = getImageUrl(part);
    }

    /**
     * Sets the profile photo URL of the mentor from a database value.
     *
     * @param profilePhotoUrl The URL of the mentor's profile photo from the database
     */
    public void setProfilePhotoUrlFromDb(String profilePhotoUrl) {
        this.profilePhotoUrl = profilePhotoUrl;
    }

```

Figure 92: Screenshot of getter and setter methods

These methods handle the retrieval and modification of various attributes of a MentorModel object. For personal details, getName(), getUsername(), getEmail(), getPhone(), and getPassword() retrieve the mentor's name, username, email, phone

number, and password, respectively. Corresponding setter methods allow for modification.

Academic details are managed by getUniversityName(), getUniversityCountry(), getMajor(), and getTuitionFee(), while getScholarship() handles scholarship information. The method getProfilePhotoUrl() retrieves the URL of the mentor's profile photo, and setProfilePhotoUrlFromPart(Part part) and setProfilePhotoUrlFromDb(String profilePhotoUrl) set the profile photo URL using either a Part object or a database value, respectively.

4.14.3. getImageUrl()

```
/*
 * Utility method to extract the image URL from a Part object.
 *
 * @param part The Part object representing the mentor's profile photo
 * @return The URL of the mentor's profile photo
 */
private String getImageUrl(Part part) {
    String savePath = StringUtils.IMAGE_DIR_USER;
    File fileSaveDir = new File(savePath);
    String imageUrlFromPart = null;
    if (!fileSaveDir.exists()) {
        fileSaveDir.mkdir();
    }

    String contentDisp = part.getHeader("content-disposition");
    String[] items = contentDisp.split(";");
    for (String s : items) {
        if (s.trim().startsWith("filename")) {
            imageUrlFromPart = s.substring(s.indexOf("=") + 2, s.length() - 1);
        }
    }

    if (imageUrlFromPart == null || imageUrlFromPart.isEmpty()) {
        imageUrlFromPart = "default-profile.png";
    }
    return imageUrlFromPart;
}
```

Figure 93: Screenshot of getImageUrl() method

This method extracts the image URL from a Part object, representing the mentor's profile photo. It checks if the directory to save the image exists, creates it if not, retrieves the filename from the content-disposition header of the Part object, and sets it as the image URL. If no filename is found or it's empty, it sets a default image URL.

4.15. SeekerModel

4.15.1. SeekerModel

```

/**
 * Default constructor for SeekerModel.
 */
public SeekerModel() {
}

/**
 * Constructs a SeekerModel with the specified parameters.
 *
 * @param name      The name of the seeker
 * @param username  The username of the seeker
 * @param email     The email of the seeker
 * @param phone     The phone number of the seeker
 * @param password  The password of the seeker
 * @param location  The location of the seeker
 * @param educationLevel The education level of the seeker
 * @param imagePart  The profile photo part of the seeker
 */
public SeekerModel(String name, String username, String email, String phone, String password, String location,
                   String educationLevel, Part imagePart) {
    super();
    this.name = name;
    this.username = username;
    this.email = email;
    this.phone = phone;
    this.password = password;
    this.location = location;
    this.educationLevel = educationLevel;
    this.profilePhotoUrl = getImageUrl(imagePart);
}

```

Figure 94: Screenshot of SeekerModel() constructor

The SeekerModel class provides two constructors for object instantiation. The default constructor initializes a new instance of SeekerModel without any parameters. On the other hand, the second constructor allows for the creation of a SeekerModel object with specific attribute values. It accepts parameters such as the seeker's name, username, email, phone number, password, location, education level, and profile photo. This constructor facilitates the creation of SeekerModel instances with predefined attribute values, providing flexibility and ease of use in the initialization process.

4.15.2. Getter and Setter methods

```
/**  
 * Returns the name of the seeker.  
 *  
 * @return The name of the seeker  
 */  
public String getName() {  
    return name;  
}  
  
/**  
 * Sets the name of the seeker.  
 *  
 * @param name The name of the seeker to set  
 */  
public void setName(String name) {  
    this.name = name;  
}  
  
/**  
 * Returns the username of the seeker.  
 *  
 * @return The username of the seeker  
 */  
public String getUsername() {  
    return username;  
}  
  
/**  
 * Sets the username of the seeker.  
 *  
 * @param username The username of the seeker to set  
 */  
public void setUsername(String username) {  
    this.username = username;  
}  
  
/**  
 * Returns the email of the seeker.  
 *  
 * @return The email of the seeker  
 */  
public String getEmail() {  
    return email;  
}  
  
/**  
 * Sets the email of the seeker.  
 *  
 * @param email The email of the seeker to set  
 */  
public void setEmail(String email) {  
    this.email = email;  
}
```

```
/**  
 * Returns the education level of the seeker.  
 *  
 * @return The education level of the seeker  
 */  
public String getEducationLevel() {  
    return educationLevel;  
}  
  
/**  
 * Sets the education level of the seeker.  
 *  
 * @param educationLevel The education level of the seeker to set  
 */  
public void setEducationLevel(String educationLevel) {  
    this.educationLevel = educationLevel;  
}  
  
/**  
 * Returns the profile photo URL of the seeker.  
 *  
 * @return The profile photo URL of the seeker  
 */  
public String getProfilePhotoUrl() {  
    return profilePhotoUrl;  
}  
  
/**  
 * Sets the profile photo URL of the seeker from the provided Part.  
 *  
 * @param part The Part containing the profile photo of the seeker  
 */  
public void setProfilePhotoUrlFromPart(Part part) {  
    this.profilePhotoUrl = getImageUrl(part);  
}  
  
/**  
 * Sets the profile photo URL of the seeker from the provided URL.  
 *  
 * @param profilePhotoUrl The profile photo URL of the seeker to set  
 */  
public void setProfilePhotoUrlFromDb(String profilePhotoUrl) {  
    this.profilePhotoUrl = profilePhotoUrl;  
}
```

```
/**  
 * Returns the phone number of the seeker.  
 *  
 * @return The phone number of the seeker  
 */  
public String getPhone() {  
    return phone;  
}  
  
/**  
 * Sets the phone number of the seeker.  
 *  
 * @param phone The phone number of the seeker to set  
 */  
public void setPhone(String phone) {  
    this.phone = phone;  
}  
  
/**  
 * Returns the password of the seeker.  
 *  
 * @return The password of the seeker  
 */  
public String getPassword() {  
    return password;  
}  
  
/**  
 * Sets the password of the seeker.  
 *  
 * @param password The password of the seeker to set  
 */  
public void setPassword(String password) {  
    this.password = password;  
}  
  
/**  
 * Returns the location of the seeker.  
 *  
 * @return The location of the seeker  
 */  
public String getLocation() {  
    return location;  
}  
  
/**  
 * Sets the location of the seeker.  
 *  
 * @param location The location of the seeker to set  
 */  
public void setLocation(String location) {  
    this.location = location;  
}
```

Figure 95: Screenshot of getter and setter methods

These methods manage the attributes of a SeekerModel object. The getters and setters handle various aspects of the seeker's information. `getName()`, `getUsername()`, `getEmail()`, `getPhone()`, `getPassword()`, `getLocation()`, and `getEducationLevel()` retrieve the seeker's name, username, email, phone number, password, location, and education level, respectively. Corresponding setter methods allow for modification. Additionally, `getProfilePhotoUrl()` retrieves the URL of the seeker's profile photo, and `setProfilePhotoUrlFromPart(Part part)` sets the profile photo URL using a Part object containing the seeker's profile photo. Lastly, `setProfilePhotoUrlFromDb(String profilePhotoUrl)` sets the profile photo URL from a provided URL.

4.15.3. `getImageUrl()`

```
/**
 * Utility method to extract the image URL from a Part object.
 *
 * @param part The Part object representing the seeker's profile photo
 * @return The URL of the seeker's profile photo
 */
private String getImageUrl(Part part) {
    String savePath = StringUtils.IMAGE_DIR_USER;
    File fileSaveDir = new File(savePath);
    String imageUrlFromPart = null;
    if (!fileSaveDir.exists()) {
        fileSaveDir.mkdir();
    }

    String contentDisp = part.getHeader("content-disposition");
    String[] items = contentDisp.split(";");
    for (String s : items) {
        if (s.trim().startsWith("filename")) {
            imageUrlFromPart = s.substring(s.indexOf("=") + 2, s.length() - 1);
        }
    }

    if (imageUrlFromPart == null || imageUrlFromPart.isEmpty()) {
        imageUrlFromPart = "default-profile.png";
    }
    return imageUrlFromPart;
}
```

Figure 96: Screenshot of `getImageUrl()` method

This method extracts the image URL from a Part object, representing the seeker's profile photo. It checks if the directory to save the image exists, creates it if not, retrieves the filename from the content-disposition header of the Part object, and sets it as the image URL. If no filename is found or it's empty, it sets a default image URL.

4.16. ServiceModel

4.16.1. ServiceModel

```

/**
 * Default constructor for ServiceModel.
 */
public ServiceModel() {
}

/**
 * Constructs a ServiceModel with the specified parameters.
 *
 * @param id           The ID of the service
 * @param title        The title of the service
 * @param description The description of the service
 * @param price        The price of the service
 * @param creationDate The creation date of the service
 * @param rating       The rating of the service
 * @param users        The number of users of the service
 * @param status       The status of the service
 * @param mentorUsername The username of the mentor providing the service
 */
public ServiceModel(int id, String title, String description, int price, LocalDate creationDate, float rating,
                    int users, String status, String mentorUsername) {
    super();
    this.id = id;
    this.title = title;
    this.description = description;
    this.price = price;
    this.creationDate = creationDate;
    this.rating = rating;
    this.users = users;
    this.status = status;
    this.mentorUsername = mentorUsername;
}

```

Figure 97: Screenshot of ServiceModel() constructor

The ServiceModel class offers two constructors for creating instances of service models. The default constructor initializes a new ServiceModel object without any parameters. Conversely, the second constructor allows for the instantiation of a ServiceModel with specific attribute values. It accepts parameters such as the service ID, title, description, price, creation date, rating, number of users, status, and mentor's username. This constructor facilitates the creation of ServiceModel instances with predefined attribute values, providing flexibility and convenience in the initialization process.

4.16.2. Getter and Setter methods

```
/**  
 * Retrieves the ID of the service.  
 *  
 * @return The ID of the service  
 */  
public int getId() {  
    return id;  
}  
  
/**  
 * Sets the ID of the service.  
 *  
 * @param id The ID of the service  
 */  
public void setId(int id) {  
    this.id = id;  
}  
  
/**  
 * Retrieves the title of the service.  
 *  
 * @return The title of the service  
 */  
public String getTitle() {  
    return title;  
}  
  
/**  
 * Sets the title of the service.  
 *  
 * @param title The title of the service  
 */  
public void setTitle(String title) {  
    this.title = title;  
}  
  
/**  
 * Retrieves the description of the service.  
 *  
 * @return The description of the service  
 */  
public String getDescription() {  
    return description;  
}  
  
/**  
 * Sets the description of the service.  
 *  
 * @param description The description of the service  
 */  
public void setDescription(String description) {  
    this.description = description;  
}
```

```
/**  
 * Retrieves the price of the service.  
 *  
 * @return The price of the service  
 */  
public int getPrice() {  
    return price;  
}  
  
/**  
 * Sets the price of the service.  
 *  
 * @param price The price of the service  
 */  
public void setPrice(int price) {  
    this.price = price;  
}  
  
/**  
 * Retrieves the creation date of the service.  
 *  
 * @return The creation date of the service  
 */  
public LocalDate getCreationDate() {  
    return creationDate;  
}  
  
/**  
 * Sets the creation date of the service.  
 *  
 * @param creationDate The creation date of the service  
 */  
public void setCreationDate(LocalDate creationDate) {  
    this.creationDate = creationDate;  
}  
  
/**  
 * Retrieves the rating of the service.  
 *  
 * @return The rating of the service  
 */  
public float getRating() {  
    return rating;  
}  
  
/**  
 * Sets the rating of the service.  
 *  
 * @param rating The rating of the service  
 */  
public void setRating(float rating) {  
    this.rating = rating;  
}
```

```

/**
 * Retrieves the number of users of the service.
 *
 * @return The number of users of the service
 */
public int getUsers() {
    return users;
}

/**
 * Sets the number of users of the service.
 *
 * @param users The number of users of the service
 */
public void setUsers(int users) {
    this.users = users;
}

/**
 * Retrieves the status of the service.
 *
 * @return The status of the service
 */
public String getStatus() {
    return status;
}

/**
 * Sets the status of the service.
 *
 * @param status The status of the service
 */
public void setStatus(String status) {
    this.status = status;
}

/**
 * Retrieves the mentor's username providing the service.
 *
 * @return The mentor's username providing the service
 */
public String getMentorUsername() {
    return mentorUsername;
}

/**
 * Sets the mentor's username providing the service.
 *
 * @param mentorUsername The mentor's username providing the service
 */
public void setMentorUsername(String mentorUsername) {
    this.mentorUsername = mentorUsername;
}

```

Figure 98: Screenshot of getter and setter methods

These methods deal with attributes of a ServiceModel object. The getters (`getId()`, `getTitle()`, `getDescription()`, `getPrice()`, `getCreationDate()`, `getRating()`, `getUsers()`, `getStatus()`, `getMentorUsername()`) retrieve specific attributes of the service. The setters (`setId()`, `setTitle()`, `setDescription()`, `setPrice()`, `setCreationDate()`, `setRating()`, `setUsers()`, `setStatus()`, `setMentorUsername()`) set the values of these attributes.

4.17. ValidationUtils

4.17.1. isAlphabetical()

```
/**
 * Validates if the provided string contains only alphabetical characters and spaces.
 *
 * @param str The string to be validated.
 * @return True if the string contains only alphabetical characters and spaces, false otherwise.
 */
public static boolean isAlphabetical(String str) {
    String pattern = "[a-zA-Z ]+";
    return str.toLowerCase().matches(pattern);
}
```

Figure 99: Screenshot of `isAlphabetical()` method

This method checks if a given string contains only alphabetical characters and spaces. It returns true if the string meets this condition, otherwise false.

4.17.2. isNumeric()

```
/**
 * Validates if the provided text contains only digits (0-9).
 *
 * @param text The text to be validated.
 * @return True if the text contains only digits, false otherwise.
 */
public static boolean isNumeric(String text) {
    return text.matches("\d+"); // Match digits only
}
```

Figure 100: Screenshot of `isNumeric()` method

This method checks if a given string contains only numbers. It returns true if the string meets this condition, otherwise false.

4.17.3. isUsername()

```
/**
 * Validates if the provided string meets the criteria for a valid username.
 *
 * @param str The username to be validated.
 * @return True if the username meets the criteria, false otherwise.
 */
public static boolean isUsername(String str) {
    String pattern = "^(?=.*[a-zA-Z])[a-zA-Z0-9_]{6,}$";
    return str.matches(pattern);
}
```

Figure 101: Screenshot of `isUsername()` method

This method checks if a given string meets the criteria for username according to pattern. It returns true if the string meets this condition, otherwise false.

4.17.4. isPhone()

```
/**  
 * Validates if the provided string is in a valid phone number format.  
 *  
 * @param str The phone number to be validated.  
 * @return True if the phone number is in a valid format, false otherwise.  
 */  
public static boolean isPhone(String str) {  
    String pattern = "^\\+[\\d{3}\\-]\\d{10}$";  
    return str.matches(pattern);  
}
```

Figure 102: Screenshot of isPhone() method

This method checks if a given string meets the criteria for phone number according to pattern. It returns true if the string meets this condition, otherwise false.

4.17.5. isPassword()

```
/**  
 * Validates if the provided string meets the criteria for a valid password.  
 *  
 * @param str The password to be validated.  
 * @return True if the password meets the criteria, false otherwise.  
 */  
public static boolean isPassword(String str) {  
    String pattern = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\\S+).{7,}$";  
    return str.matches(pattern);  
}
```

Figure 103: Screenshot of isPassword() method

This method checks if a given string meets the criteria for password according to pattern. It returns true if the string meets this condition, otherwise false.

4.17.6. isImage()

```
/**  
 * Checks if the provided string represents a valid image file based on its extension.  
 *  
 * @param str The string representing the image file name.  
 * @return True if the string represents a valid image file, false otherwise.  
 */  
public static boolean isImage(String str) {  
    return str.endsWith(".jpg") || str.endsWith(".jpeg") || str.endsWith(".png") ||  
        str.endsWith(".gif") || str.endsWith(".bmp") || str.endsWith(".tif") ||  
        str.endsWith(".tiff") || str.endsWith(".svg");  
}
```

Figure 104: Screenshot of isImage() method

This method checks if a given string meets the criteria for image suffix. It returns true if the string meets this condition, otherwise false.

4.17.7. isEmail()

```
/**  
 * Validates if the provided text is a valid email address format.  
 *  
 * @param email The email address to be validated.  
 * @return True if the email address has a valid format, false otherwise.  
 */  
public static boolean isEmail(String email) {  
    return email.matches("^[\\w-\\.]+@[\\w-\\.]+[\\w-]{2,}$"); // Match standard email pattern  
}
```

Figure 105: Screenshot of isEmail() method

This method checks if a given string meets the criteria for email according to pattern. It returns true if the string meets this condition, otherwise false.

5. Database

4.1. Assumptions

- ◆ A mentor can provide many services.
- ◆ A service can be used by one or more seekers.
- ◆ A service cannot exist without a mentor.
- ◆ Mentor's and Seeker's username are all unique.
- ◆ A service can be uniquely identified by their id.

4.2. Normalization

a. Unnormalized Form (UNF)

The Table in UNF is:

Mentor (Mentor Username, Mentor Name, Mentor Phone, Mentor Email, Mentor Password, Mentor Photo, University Name, University Country, Major, Tuition Fee, Scholarship, { Service ID, Service Title, Description, Price, Creation Date, Rating, Service Users, Status { Seeker Username, Seeker Name, Seeker Email, Seeker Password, Seeker Phone, Seeker Location, Seeker Photo, Education Level }})

b. First Normal Form (1NF)

The Tables in 1NF are:

Mentor-1 { Mentor Username (P.K.), Mentor Name, Mentor Phone, Mentor Email, Mentor Password, Mentor Photo, University Name, University Country, Major, Tuition Fee, Scholarship }

Service-1 { Service ID(P.K.), Mentor Username(F.K.), Service Title, Description, Price, Creation Date, Rating, Service Users, Status }

Seeker-1 { Seeker Username(P.K.), Mentor Username(F.K.), Service ID(F.K.), Seeker Name, Seeker Email, Seeker Password, Seeker Phone, Seeker Location, Seeker Photo, Education Level }

c. Second Normal Form (2NF)

Mentor -1 table has only one key attribute, so it remains the same in second normal form.

For Service-1 Table (Two key columns), checking for partial dependencies for $(2^2 - 1) = 3$ combinations :

Service ID --> X

Mentor Username --> X

Service ID, Mentor Username --> Service Title, Description, Price, Creation Date, Rating, Service Users, Status

Similarly, For Seeker-1 Table (Three key columns), checking for partial dependencies for $(2^3 - 1) = 7$ combinations:

Seeker Username --> Seeker Name, Seeker Email, Seeker Phone, Seeker Password, Seeker Location, Seeker Photo, Education Level

Mentor Username --> X

Service ID --> X

Seeker Username, Mentor Username --> X

Service ID, Seeker Username --> X

Mentor Username, Service ID --> X

Seeker Username, Mentor Username, Service ID --> X

The Tables in 2NF are:

Mentor-2 { Mentor Username (P.K.), Mentor Name, Mentor Phone, Mentor Email, Mentor Password, Mentor Photo, University Name, University Country, Major, Tuition Fee, Scholarship }

Seeker-2 { Seeker Username(P.K.), Seeker Name, Seeker Email, Seeker Password, Seeker Phone, Seeker Location, Seeker Photo, Education Level }

Service-2 { Service ID(P.K.), Mentor Username(F.K.), Service Title, Description, Price, Creation Date, Rating, Service Users, Status }

Service-Details-2 { Service ID (F.K.), Seeker Username (F.K.) }

The rest of the unwanted bridge tables are removed.

d. Third Normal Form (3NF)

None of the tables in 2NF have transitive dependencies so they remain the same.

Hence,

The Tables in 3NF are:

- Mentor-3 { Mentor Username (P.K.), Mentor Name, Mentor Phone, Mentor Email, Mentor Password, Mentor Photo, University Name, University Country, Major, Tuition Fee, Scholarship }
- Service-3 { Service ID(P.K.), Mentor Username(F.K.), Service Title, Description, Price, Creation Date, Rating, Service Users, Status }
- Seeker-3 { Seeker Username(P.K.), Seeker Name, Seeker Email, Seeker Password, Seeker Phone, Seeker Location, Seeker Photo, Education Level }
- Service-Details-3 { Service ID (F.K.), Seeker Username (F.K.) }

4.2. Final Entity Relationship Diagram (ERD)

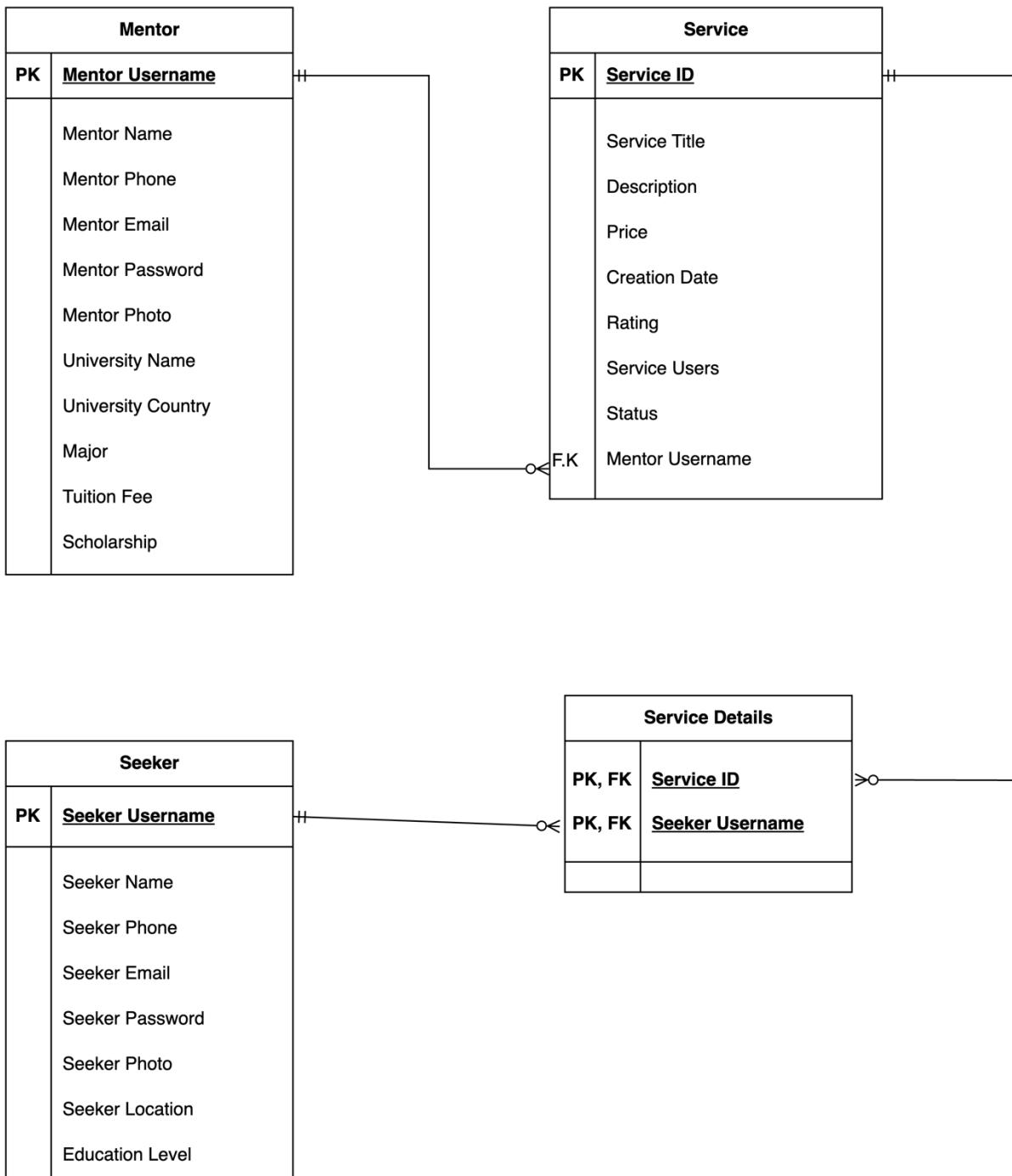


Figure 106: Final Entity Relationship Diagram

4.3. Database design

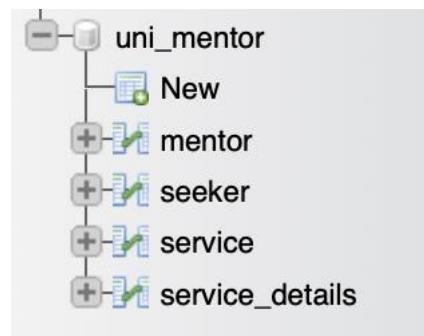


Figure 107: Database and database tables

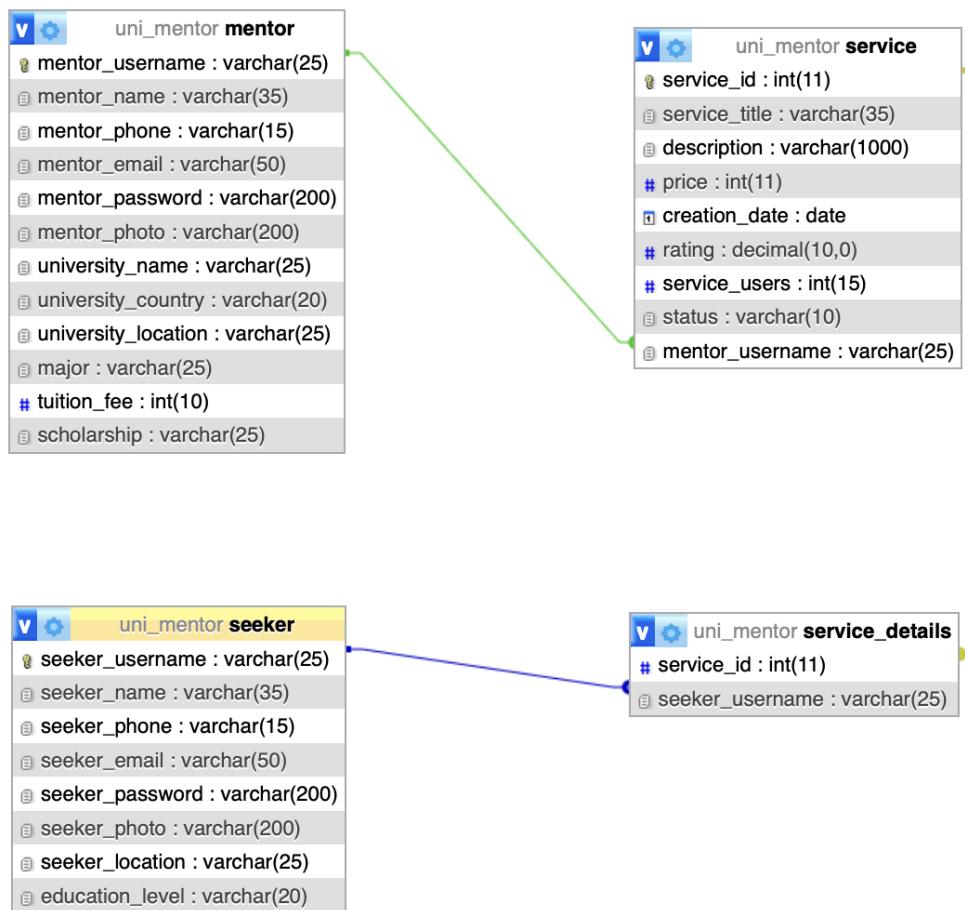


Figure 108: Database design

4.4. Table design

i. Mentor Table:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	mentor_username	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More
2	mentor_name	varchar(35)	utf8mb4_general_ci		No	None			Change Drop More
3	mentor_phone	varchar(15)	utf8mb4_general_ci		No	None			Change Drop More
4	mentor_email	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
5	mentor_password	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
6	mentor_photo	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
7	university_name	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More
8	university_country	varchar(20)	utf8mb4_general_ci		No	None			Change Drop More
9	major	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More
10	tuition_fee	int(10)			No	None			Change Drop More
11	scholarship	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More

Figure 109: Mentor table

ii. Seeker Table:

Table structure		Relation view							
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	seeker_username	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More
2	seeker_name	varchar(35)	utf8mb4_general_ci		No	None			Change Drop More
3	seeker_phone	varchar(15)	utf8mb4_general_ci		No	None			Change Drop More
4	seeker_email	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
5	seeker_password	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
6	seeker_photo	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More
7	seeker_location	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More
8	education_level	varchar(20)	utf8mb4_general_ci		No	None			Change Drop More

Figure 110: Seeker Table

iii. Service Table:

Table structure		Relation view							
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	service_id	int(11)			No	None	AUTO_INCREMENT		Change Drop More
2	service_title	varchar(35)	utf8mb4_general_ci		No	None			Change Drop More
3	description	varchar(1000)	utf8mb4_general_ci		No	None			Change Drop More
4	price	int(11)			No	None			Change Drop More
5	creation_date	date			No	None			Change Drop More
6	rating	decimal(10,1)			No	None			Change Drop More
7	service_users	int(15)			No	None			Change Drop More
8	status	varchar(10)	utf8mb4_general_ci		No	None			Change Drop More
9	mentor_username	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More

Figure 111: Service Table

iv. Service-Details Table:

The screenshot shows the 'Table structure' view in MySQL Workbench. The table has two columns: 'service_id' and 'seeker_username'. The 'service_id' column is of type int(11) and the 'seeker_username' column is of type varchar(25). Both columns have a collation of utf8mb4_general_ci, are not nullable (No), and have a default value of None. There are edit, drop, and more options available for each column.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	service_id	int(11)			No	None			Change Drop More
2	seeker_username	varchar(25)	utf8mb4_general_ci		No	None			Change Drop More

Figure 112: Service-details table

5. Test cases

5.1. Test 1: Test for Register (mentor & seeker)

Test 1.1. : Test for input register details validation

Objective	To display error message when invalid details are entered for registration.
Action	<ul style="list-style-type: none"> ➤ Invalid details are entered in register-mentor and register-seeker pages. ➤ Register button is clicked.
Expected Result	Error message indicating error input should be displayed.
Actual Result	Error message was displayed.
Conclusion	The test is successful.

Table 1: Test table for register details validation

Proofs:

- 1) Entering invalid details in register-mentor page:

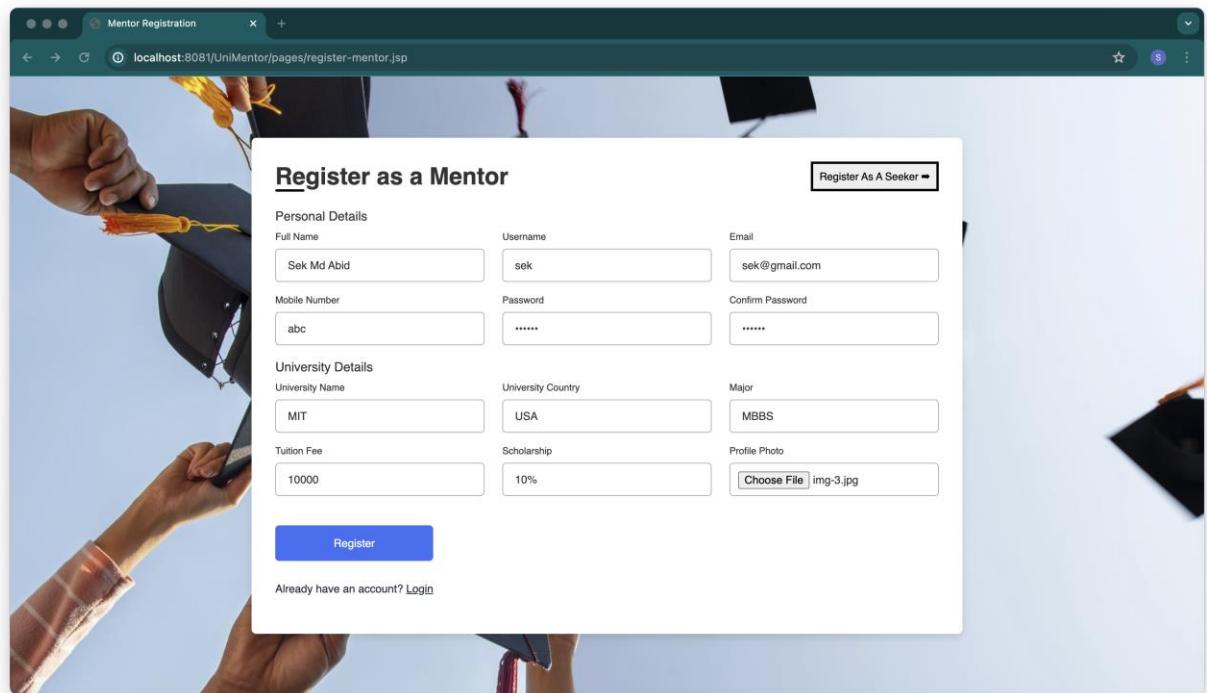


Figure 113: Screenshot of entering invalid details

2) Register button is clicked:

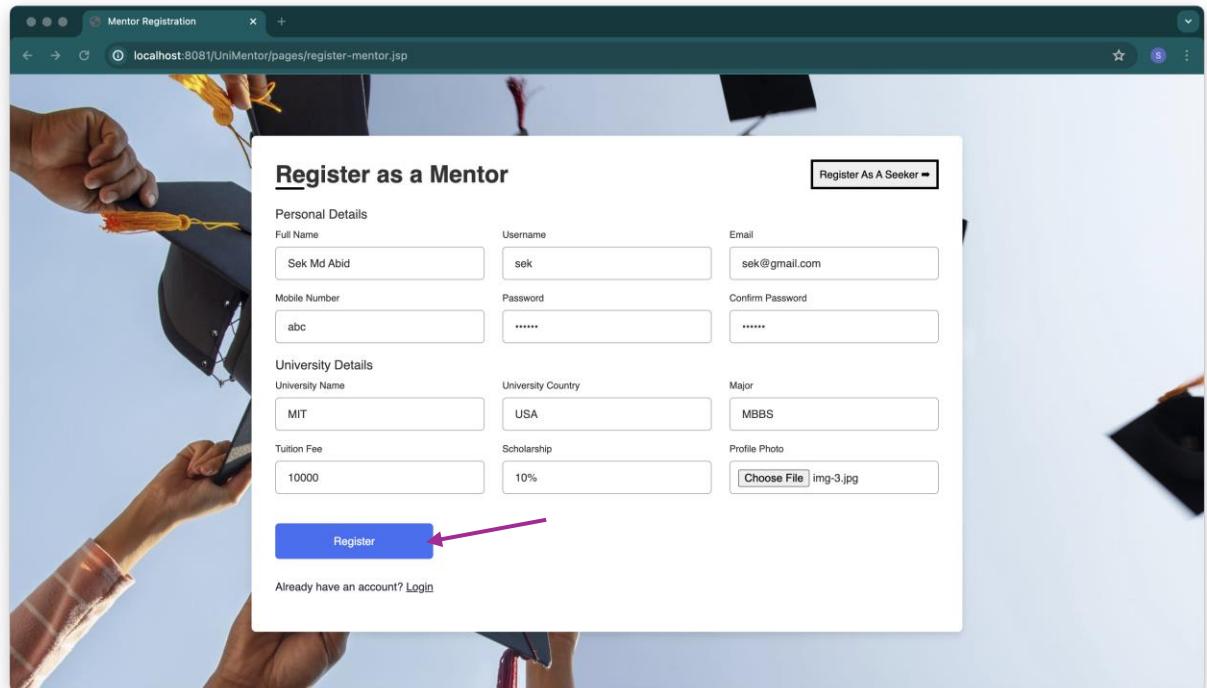
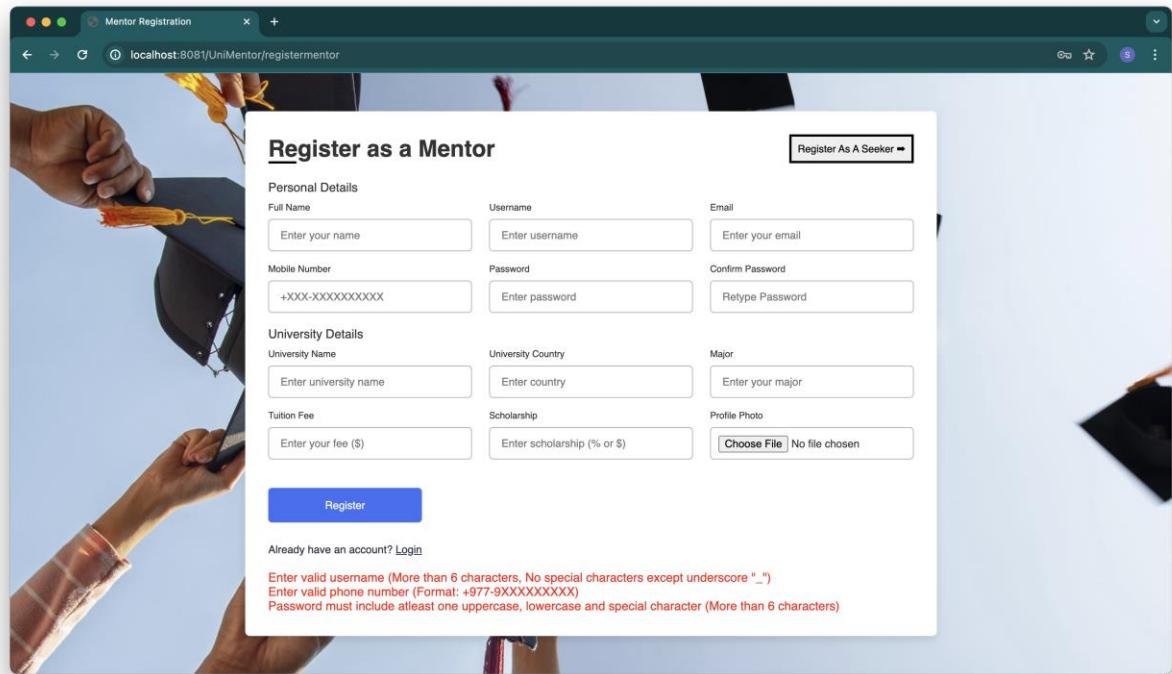


Figure 114: Screenshot of register button clicked

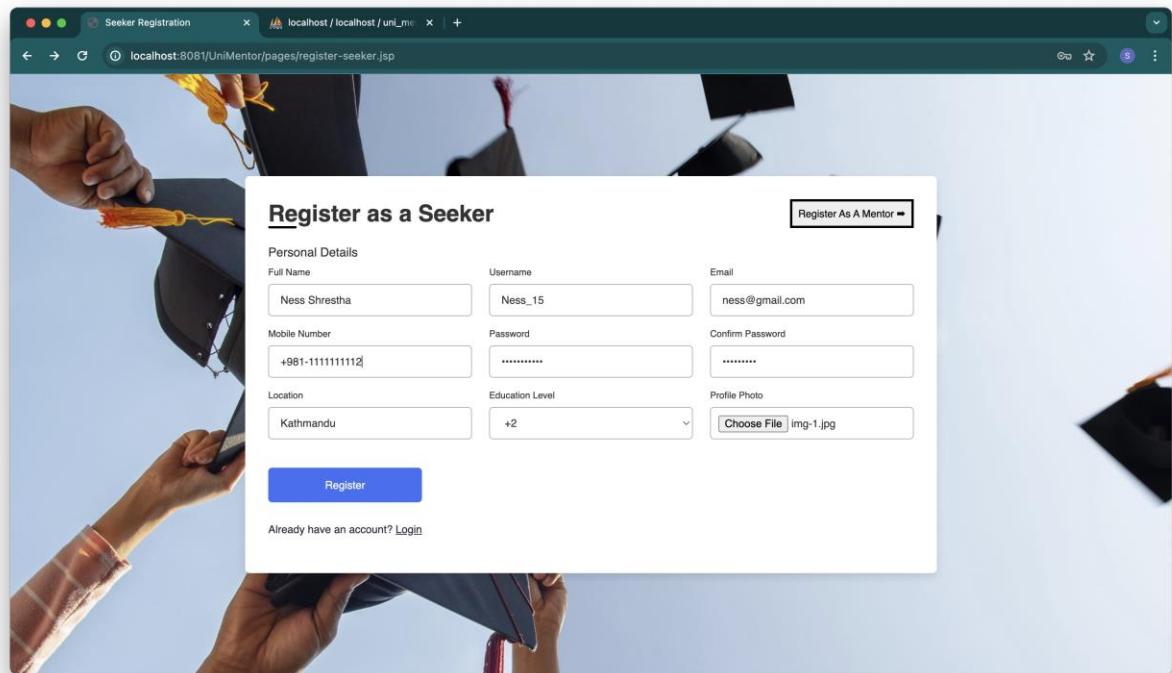
3) Error message displayed:



The screenshot shows a registration form titled "Register as a Mentor" on a web browser. The form includes fields for Personal Details (Full Name, Username, Email, Mobile Number, Password, Confirm Password), University Details (University Name, University Country, Major, Tuition Fee, Scholarship, Profile Photo), and a "Register" button. A "Register As A Seeker" link is also present. Below the form, there is a message: "Already have an account? [Login](#)". At the bottom, three validation errors are displayed in red: "Enter valid username (More than 6 characters, No special characters except underscore '_')", "Enter valid phone number (Format: +977-XXXXXX XXXXX)", and "Password must include atleast one uppercase, lowercase and special character (More than 6 characters)".

Figure 115: Screenshot of error message displayed

4) Entering invalid details in register-seeker page:



The screenshot shows a registration form titled "Register as a Seeker" on a web browser. The form includes fields for Personal Details (Full Name, Username, Email, Mobile Number, Password, Confirm Password), Location (Location, Education Level), and a "Profile Photo" section where a file named "img-1.jpg" is selected. A "Register" button is at the bottom. A "Register As A Mentor" link is also present. Below the form, there is a message: "Already have an account? [Login](#)".

Figure 116: Screenshot of details entered in register seeker

5) Register button is clicked:

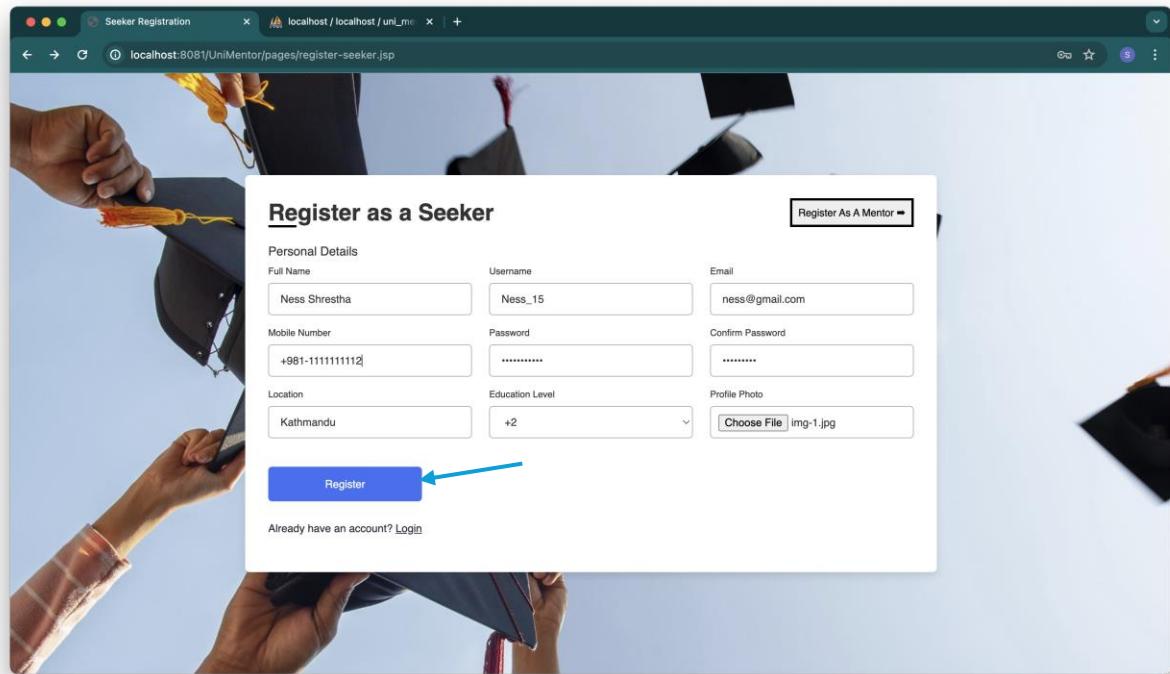


Figure 117: Screenshot of register button clicked

6) Error message displayed:

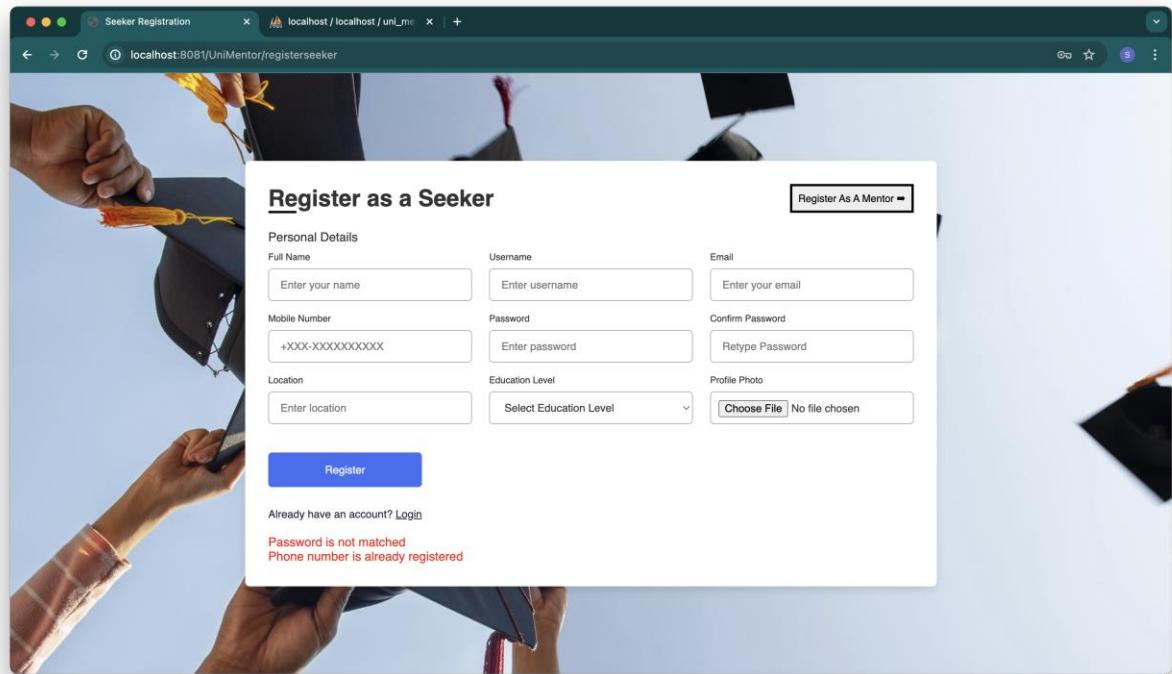


Figure 118: Screenshot of Error message displayed

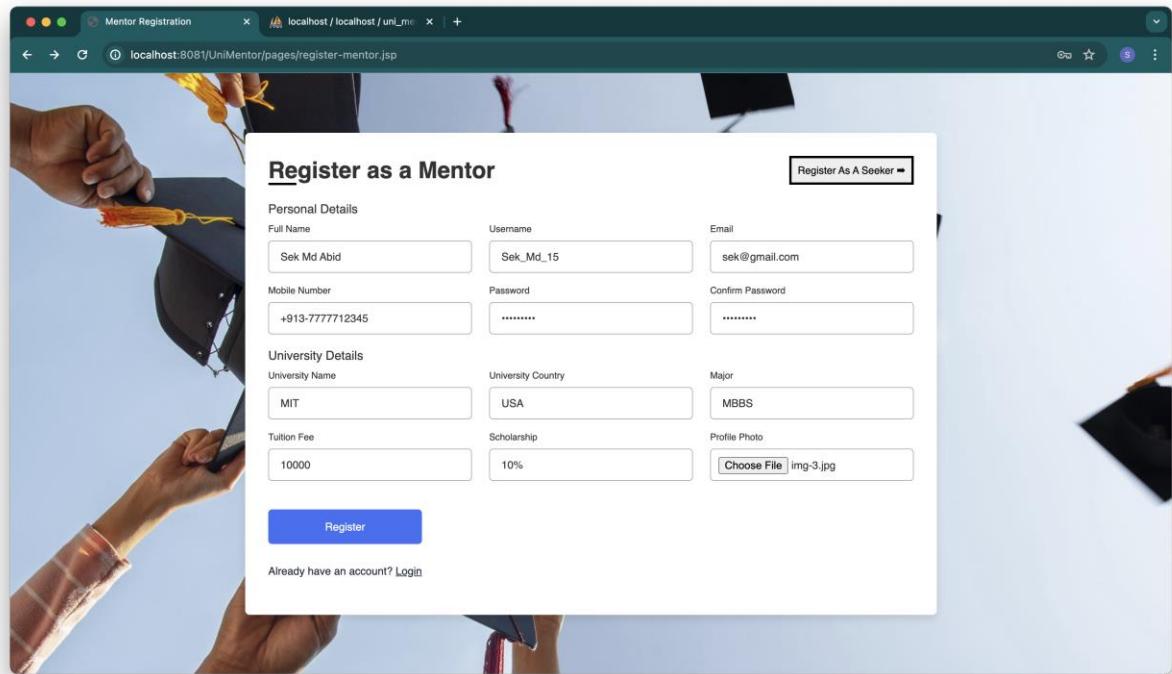
Test 1.2. : Test for successful registration

Objective	To display success message when registration process is successful.
Action	<ul style="list-style-type: none"> ➤ Correct details are entered in register-mentor and register-seeker pages. ➤ Register button is clicked.
Expected Result	Success message indicating successful registration should be displayed.
Actual Result	Success message was displayed.
Conclusion	The test is successful.

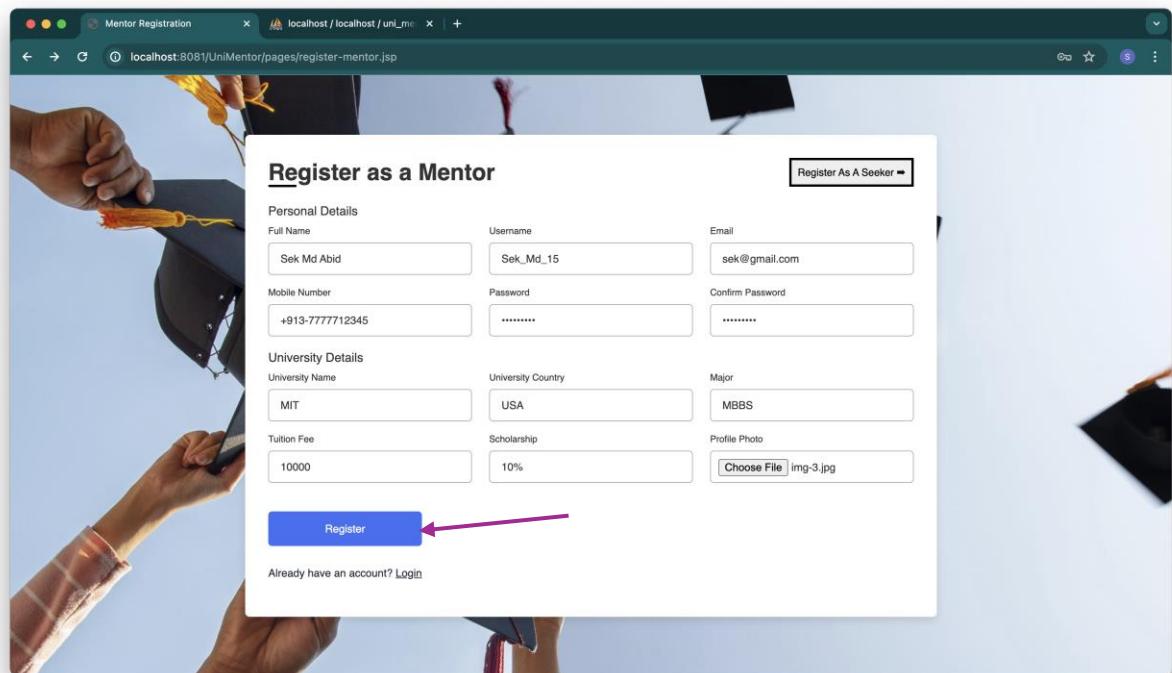
Table 2: Test table for success reigstration

Proofs:

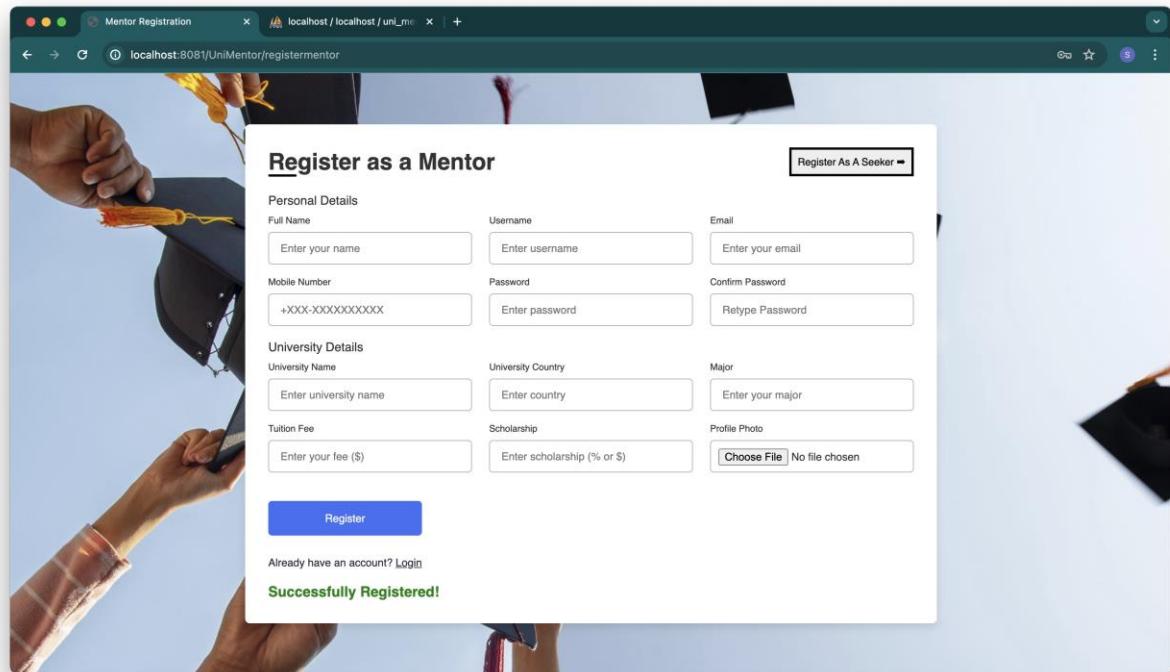
- 1) Entering correct details in register-mentor page:



2) Register button is clicked:



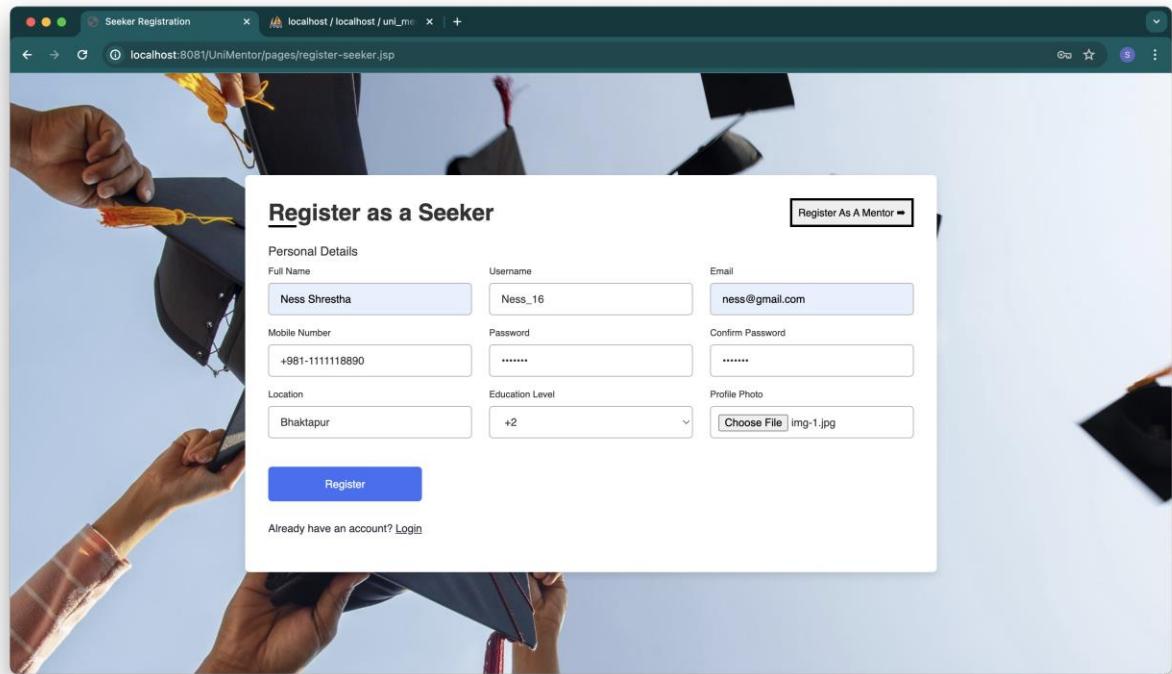
3) Success message displayed:



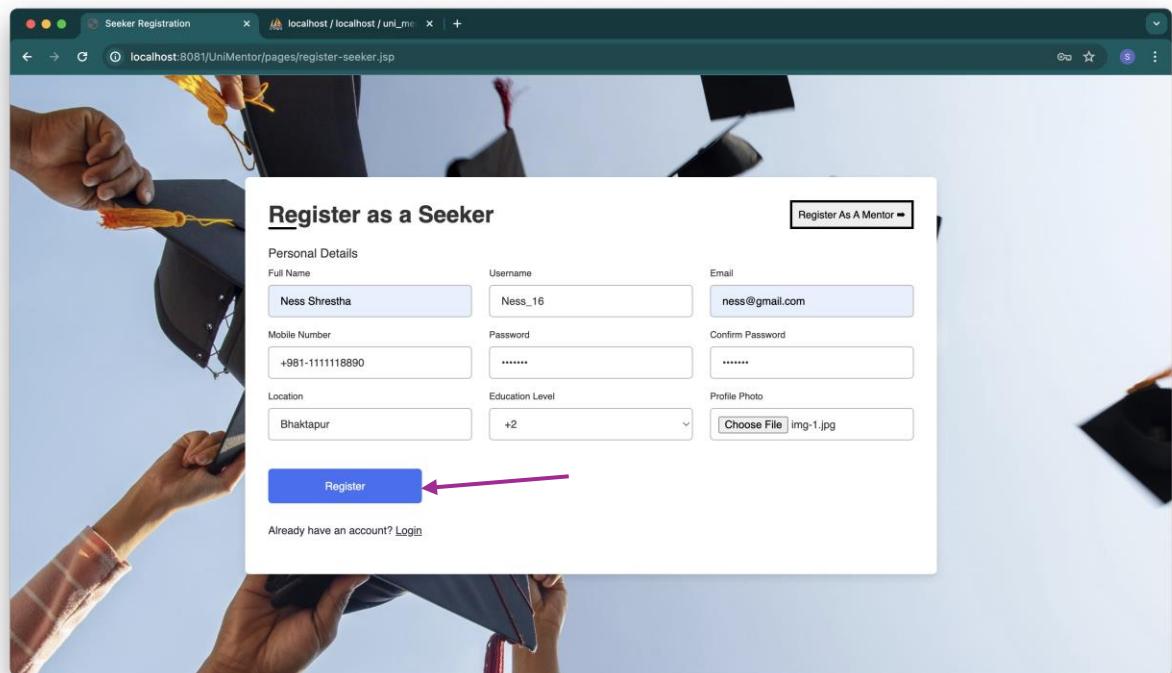
4) Mentor registered in database

	mentor_username	mentor_name	mentor_phone	mentor_email	mentor_password	mentor_photo	university_name
<input type="checkbox"/>	Edit	Sek Md Abid	+91-7777712345	sek@gmail.com	GswrmN5zDmIOh7PrZOlx4BpnumAh7ySpug9UkFDHoTbAFrX4wn...	img-3.jpg	MIT
<input type="checkbox"/>	Edit	Sardul_Ojha_11	+977-9742855801	sardul@gmail.com	NPGK7AE5Uljn4M4b1YgRnb84qHsHu01MMtwM2yQ3zm3uNzKW...	img-2.jpg	Harvard Univers
<input type="checkbox"/>	Edit	Hridaya_12	+91-8888888888	hir@hotmail.com	SvdnJ936W7Y1jTDx1YmOvL8j7D8T4pi++eDMzGsTVJvSkS84...	default-profile.png	Stanford Univers

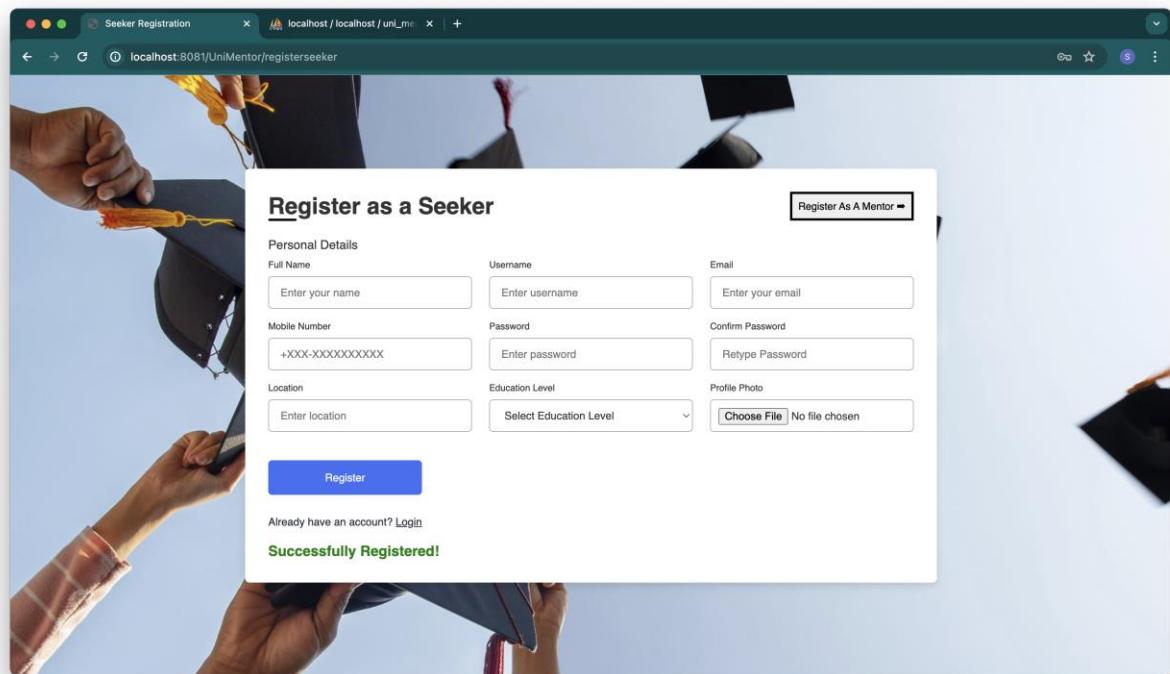
5) Entering correct details in register-seeker page:



6) Register button is clicked:



7) Success message displayed:



8) Seeker Registered in database:

Showing rows 0 - 2 (3 total, Query took 0.0001 seconds.)

SELECT * FROM `seeker`

	seeker_username	seeker_name	seeker_phone	seeker_email	seeker_password	seeker_photo	seeker_location
<input type="checkbox"/>	Edit	Copy	Delete	Ness_16	Ness Shrestha +981- 1111118890	ness@gmail.com I3nferWslXDr6/Ze2ZKwPD3FCMaMCr3h5gMrkAYsxGhbvUfY... img-1.jpg	Bhaktapur
<input type="checkbox"/>	Edit	Copy	Delete	Ronak_13	Ronak Shrestha +977- 8888888831	ronak@yahoo.com ANwfRl6Tc3FMqGwvBh9VQZ1DhX3dZuG1WHgCl77QF04D7gu1U... img-4.jpg	Kathmandu, Nepa
<input type="checkbox"/>	Edit	Copy	Delete	Suyash_14	Suyash Dhakal +981-111111112	suyash@haha.com agTdBt32HxvwirZY44UxRT8hpDNx0clpqyz3Fc4ejpnAkdy/FT... default-profile.png	Beijing, China

With selected: Edit Copy Delete Export

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

Console

5.2. Test 2: Test for Login

Test 2.1. : Test for input login details validation

Objective	To display error message when invalid details are entered for login.
Action	<ul style="list-style-type: none"> ➤ Invalid login details are entered in login page. ➤ Login button is clicked.
Expected Result	Error message indicating error login details should be displayed.
Actual Result	Error message was displayed.
Conclusion	The test is successful.

Table 3: Test table for login details validation

Proofs:

- 1) Entering invalid details in login page:

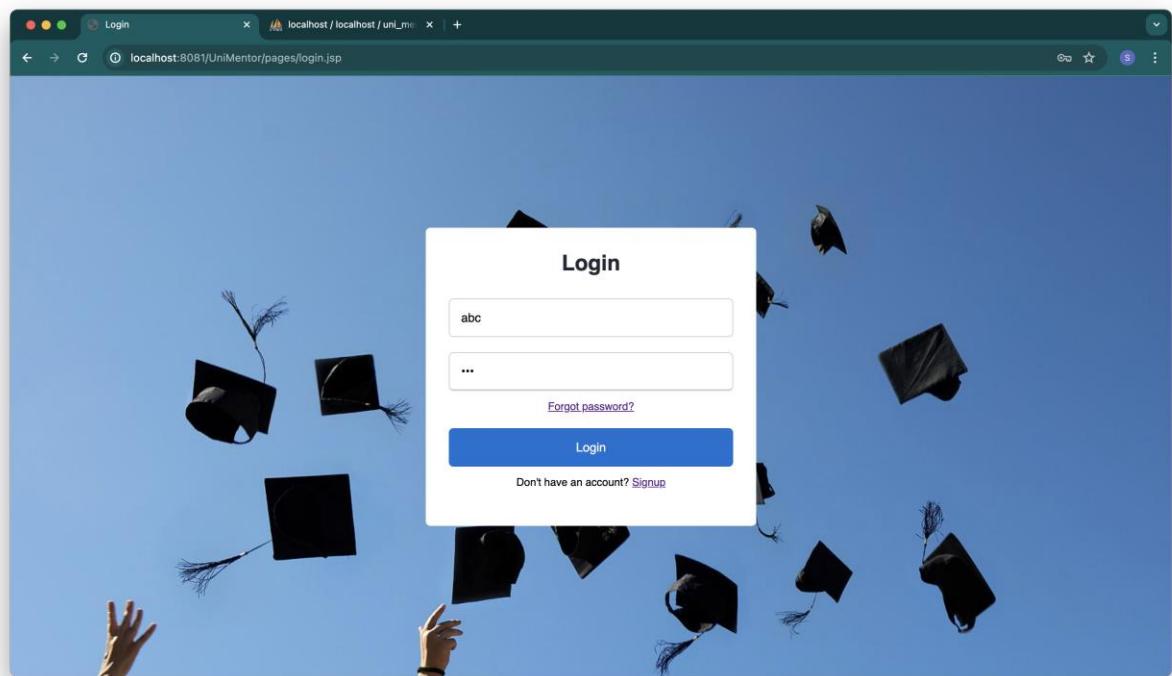


Figure 119: Screenshot of entering invalid details

- 2) Login button is clicked:

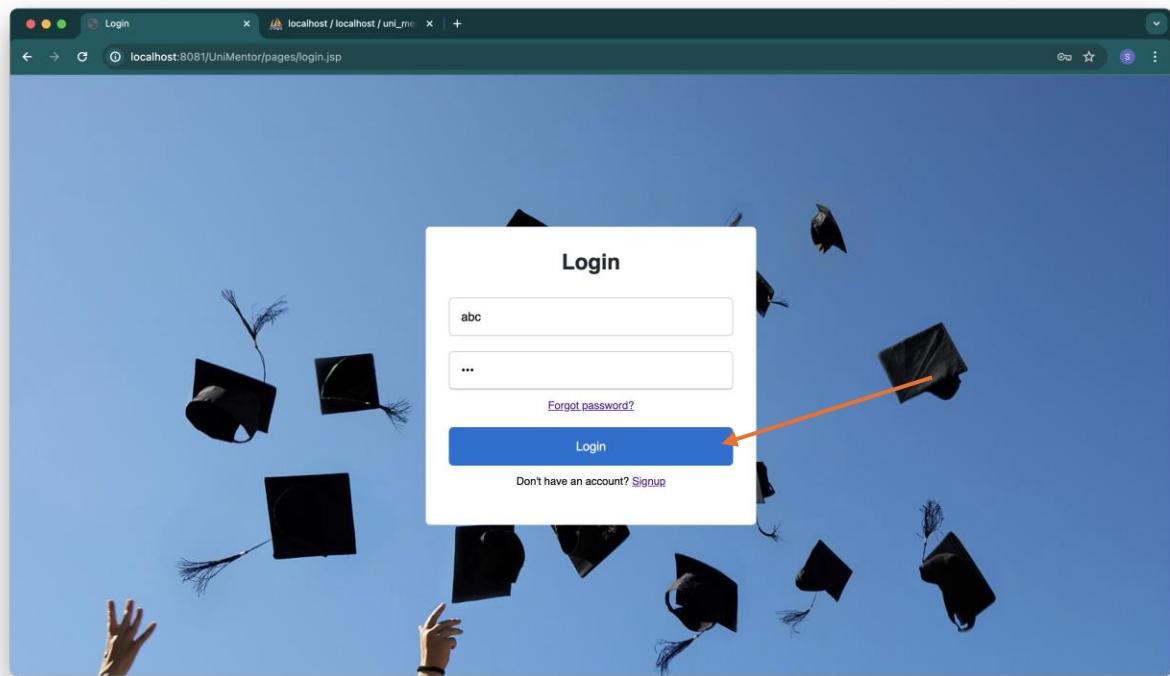


Figure 120: Screenshot of login button clicked

3) Error message displayed:

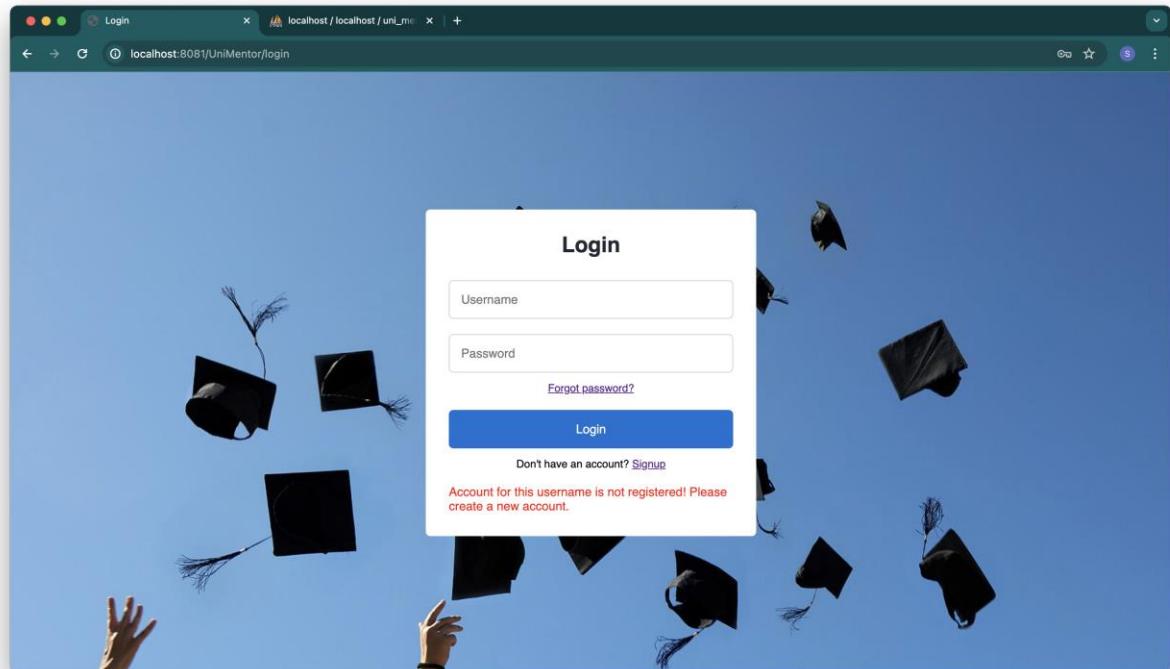


Figure 121: Screenshot of error message displayed

Test 2.2. : Test for successful login

Objective	To login successful when valid username and password is entered.
Action	<ul style="list-style-type: none"> ➤ Correct login details are entered in login page. ➤ Login button is clicked.
Expected Result	Welcome page welcoming the user should be displayed.
Actual Result	The user was navigated to welcome page.
Conclusion	The test is successful.

Table 4: Test table for login success

Proofs:

- 1) Entering valid login details in login page:

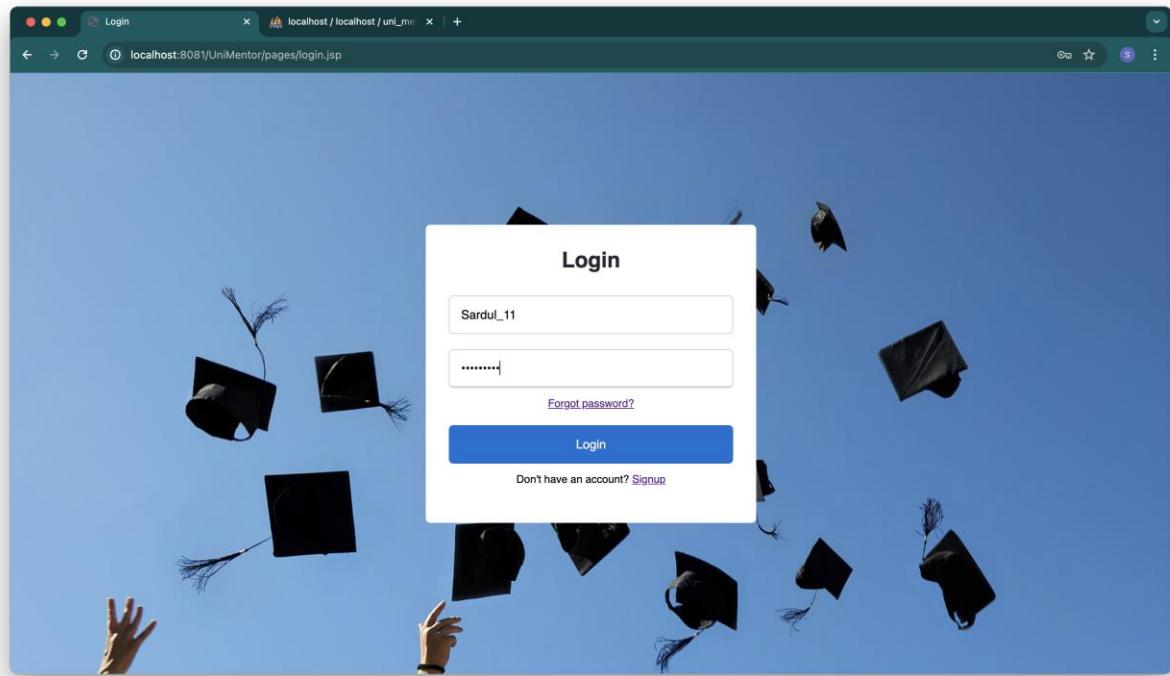


Figure 122: Screenshot of entering valid login details

- 2) Login button is clicked:

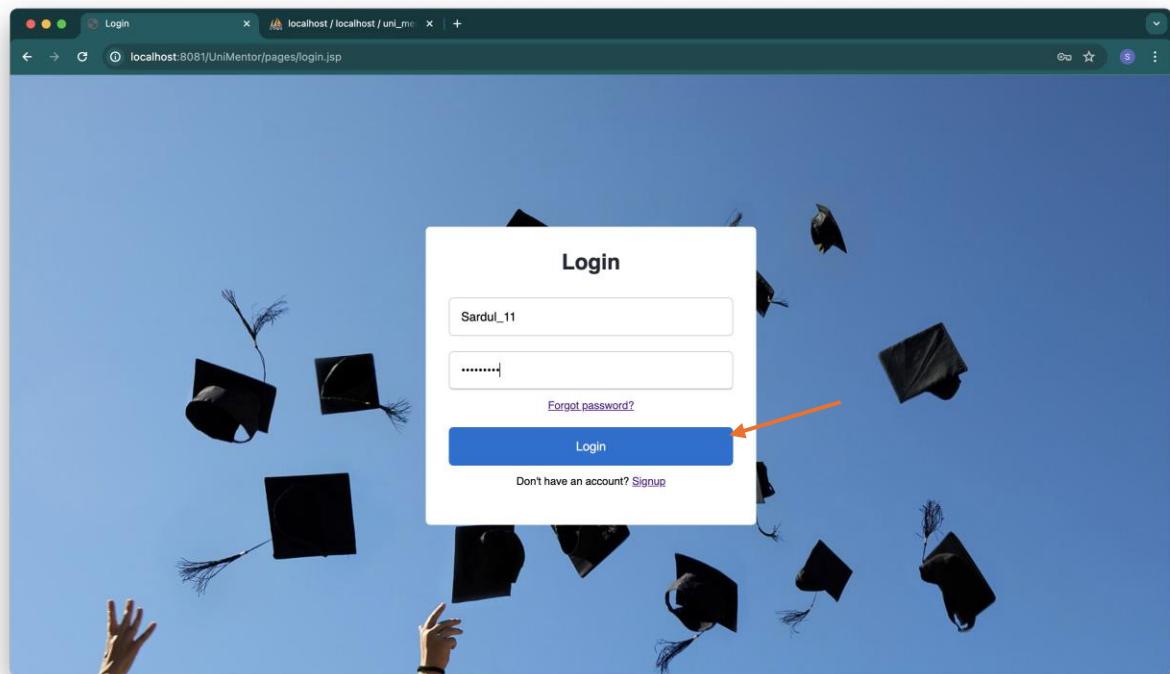


Figure 123: Screenshot of login button clicked

3) Welcome page displayed:

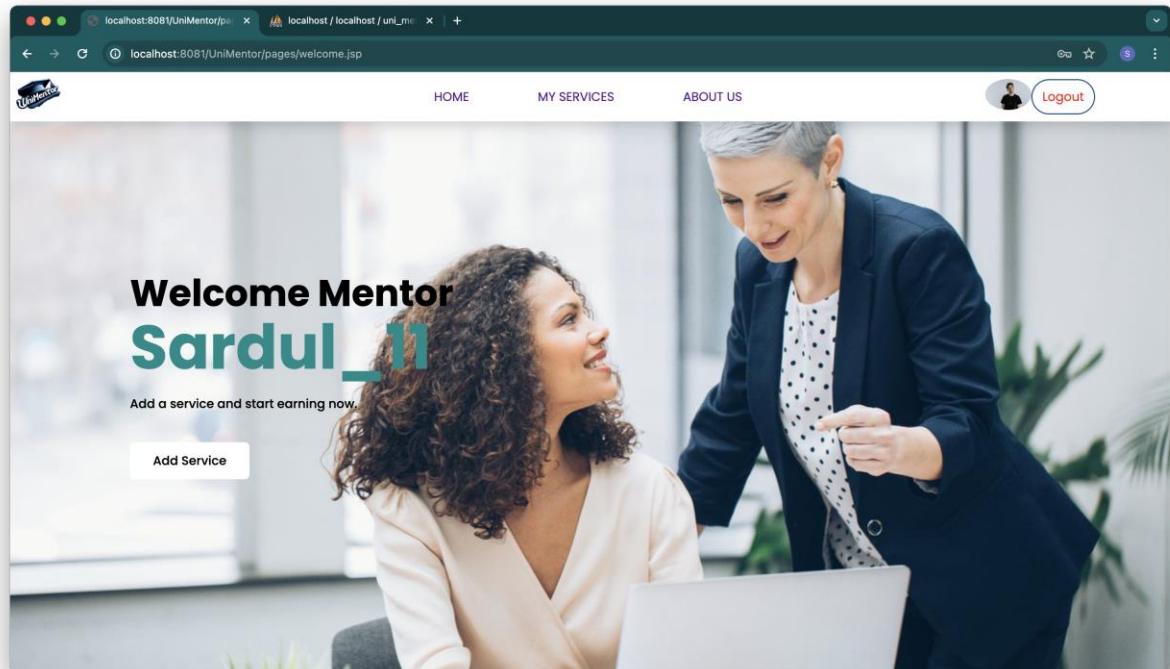


Figure 124: Screenshot of welcome page displayed

5.3. Test 3: Test for Display

Test 3.1. : Test for all services displayed for seekers

Objective	To all the services from database for seeker.
Action	<ul style="list-style-type: none"> ➤ Logged in as a seeker. ➤ Service is clicked in nav bar to navigate to services page.
Expected Result	All the services in the database should be displayed.
Actual Result	All the services were displayed.
Conclusion	The test is successful.

Table 5: Test table for service display to seekers

Proofs:

- 1) Logging in as a seeker:

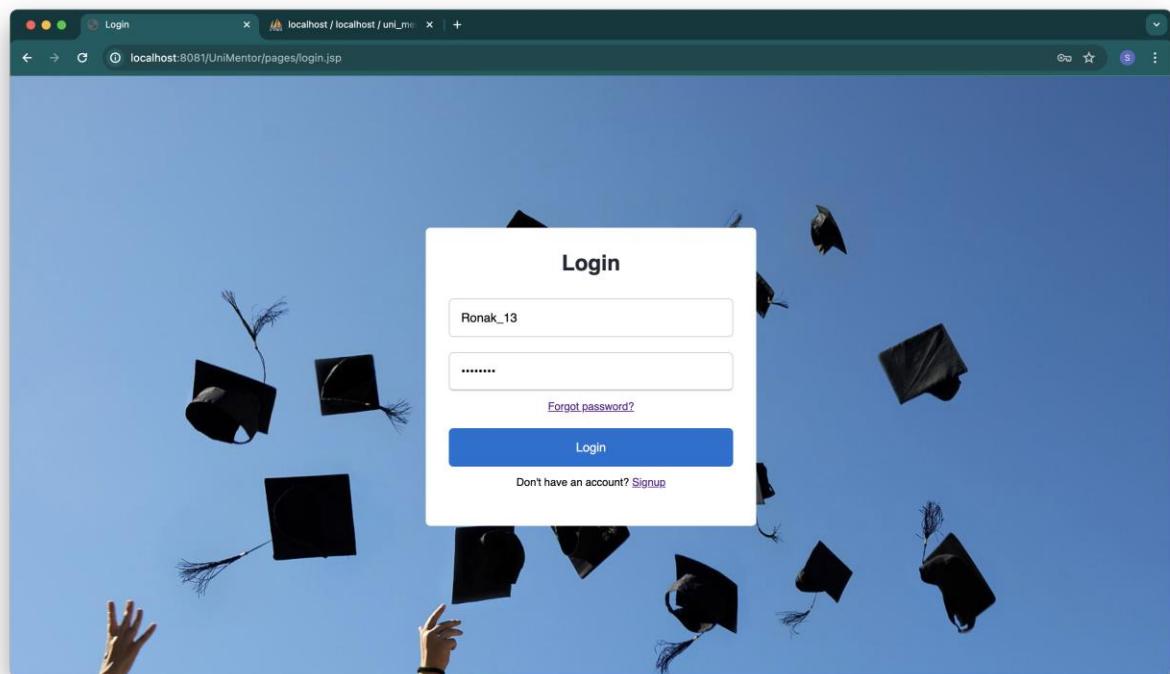


Figure 125: Screenshot of entering seeker login details

2) Clicking service button nav bar to navigate to services page:

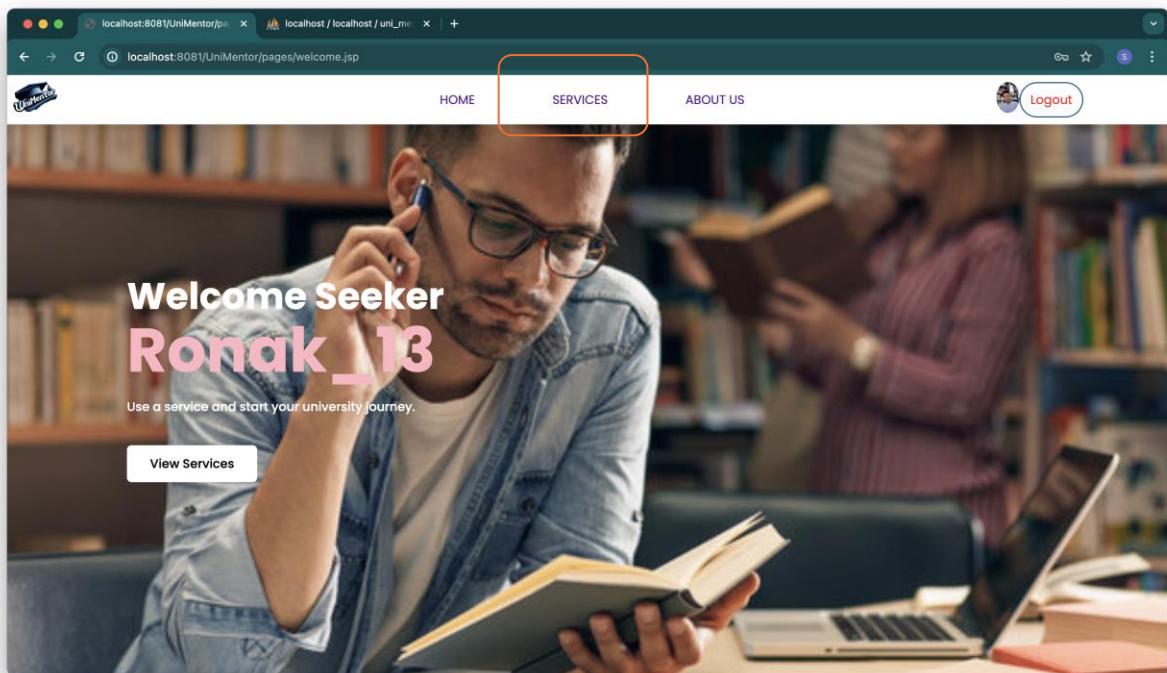


Figure 126: Screenshot of clicking service button

3) All services present in database

The screenshot shows the phpMyAdmin interface for the 'uni_mentor' database. The left sidebar shows the database structure with tables like 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'student_management', and 'uni_mentor'. The 'uni_mentor' table contains three rows of data:

service_id	service_title	description	price	creation_date	rating	service_users	status	mentor_username
72	Interview Preparation Package	Nail your university interviews with my comprehens...	150	2024-05-10	3.0	289	Active	Hridaya_12
73	Full Application Process Guide	Get expert guidance for your university applicatio...	200	2024-05-10	4.3	19	Active	Sardul_11
74	Writing Personal Statement	Struggling with your university personal statement...	100	2024-05-10	4.5	56	Active	Sardul_11

Figure 127: Screenshot of all services in database

4) All the services displayed:

The screenshot shows the 'All Services' page from the UniMentor application. The top navigation bar includes links for 'HOME', 'SERVICES', and 'ABOUT US', along with a user profile icon and a 'Logout' button. Below the navigation is a search bar with placeholder text 'Search by Service Name...' and a magnifying glass icon.

The main content area is titled 'All Services' and displays three service cards:

- Sardul Ojha** (Harvard University, 100% Scholarship)
 - Writing Personal Statement**: Struggling with your university personal statement? I'm here to help! With personalized guidance and attention to detail, I'll work with you to create a standout statement that reflects your strengths and ambitions. Let's make your application stand out
 - \$100, 4.5 ⭐, 56 Users
 - [View Details](#), [Heart](#)
- Sardul Ojha** (Harvard University, 100% Scholarship)
 - Full Application Process Guide**: Get expert guidance for your university application journey with my Full Application Process Guide. From personal statements to interviews, I'll help you every step of the way. Let's make your application a success reach out today!
 - \$200, 4.3 ⭐, 19 Users
 - [View Details](#), [Heart](#)
- Hridaya Giri** (Stanford University, 80% Scholarship)
 - Interview Preparation Package**: Nail your university interviews with my comprehensive Interview Preparation Package. Gain confidence, learn effective strategies, and practice with mock interviews tailored to your chosen institutions. Let's ensure your interview performance shines!
 - \$150, 3.0 ⭐, 289 Users
 - [View Details](#), [Heart](#)

Figure 128: Screenshot of all service displayed in service page

Test 3.2. : Test for own services displayed for mentors

Objective	To display only current mentor's services from database.
Action	<ul style="list-style-type: none"> ➤ Logged in as a mentor. ➤ My service is clicked in nav bar.
Expected Result	Only the logged in mentor's services should be displayed with the option to add, update, and delete.
Actual Result	The services were displayed.
Conclusion	The test is successful.

Table 6: Test table for service display to mentors

Proofs:

- 1) Logging in as a mentor:

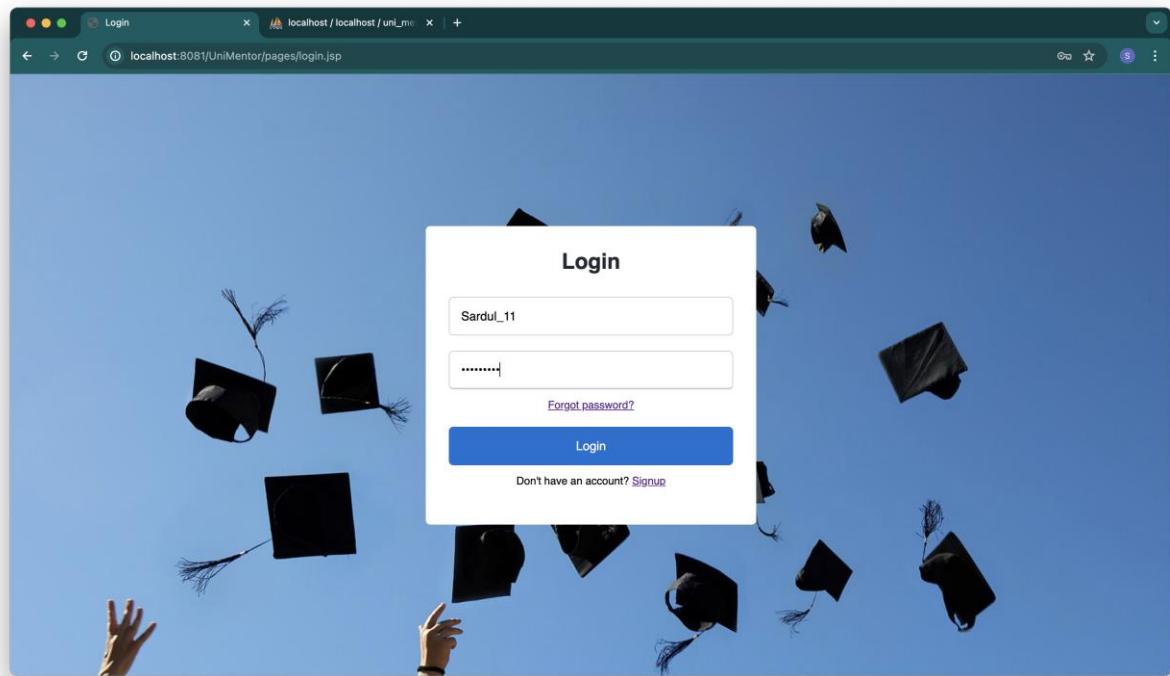


Figure 129: Screenshot of entering login details as mentor

- 2) Clicking my service button nav bar to navigate to my services page:

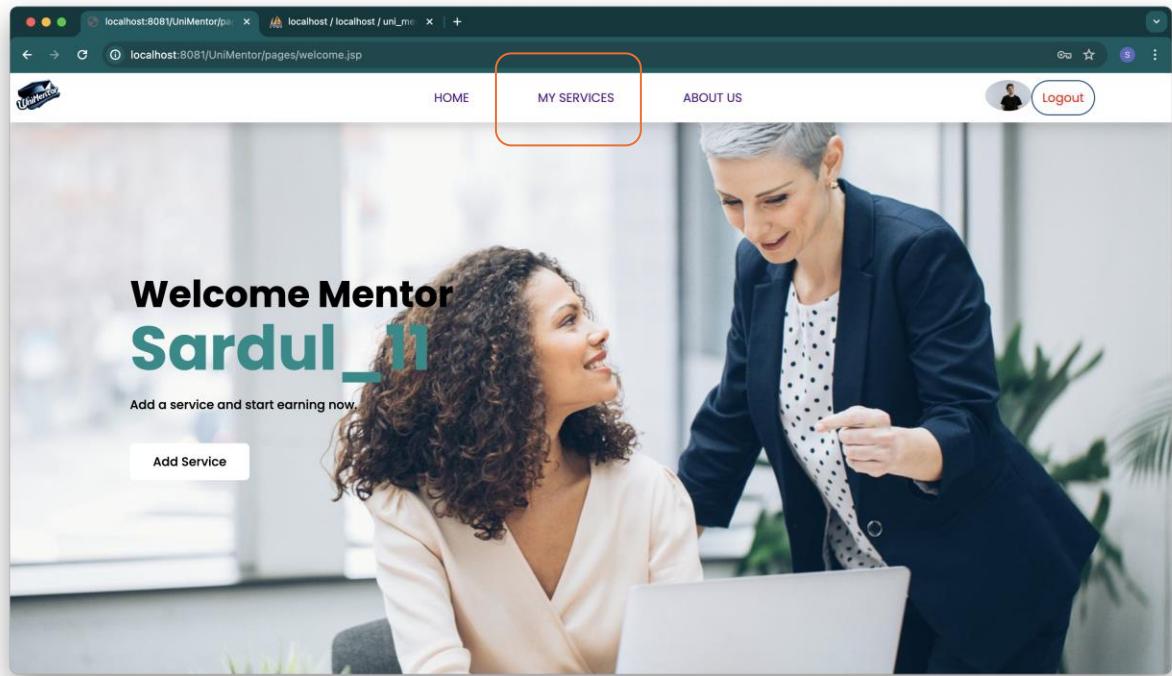


Figure 130: Screenshot of clicking my service button in nav

3) All the logged in mentor's services displayed:

Service Name	Description	Price	Rating	Users
Writing Personal Statement	Struggling with your university personal statement? I'm here to help! With personalized guidance and attention to detail, I'll work with you to create a standout statement that reflects your strengths and ambitions. Let's make your application stand out	\$ 100	4.5 ⭐	56 Users
Full Application Process Guide	Get expert guidance for your university application journey with my Full Application Process Guide. From personal statements to interviews, I'll help you every step of the way. Let's make your application a success reach out today!	\$ 200	4.3 ⭐	19 Users

Figure 131: Screenshot of mento's own service displayed

5.4. Test 4: Test for Add

Test 4.1. : Test for new service being added by mentor

Objective	To add a new service.
Action	<ul style="list-style-type: none"> ➤ Add service button is clicked in my services. ➤ Details of service to be added are entered. ➤ Add button is clicked.
Expected Result	The added service should be displayed in my services.
Actual Result	The added service was displayed.
Conclusion	The test is successful.

Table 7: Test table for new service added

Proofs:

- 1) Clicking add service button in my services:

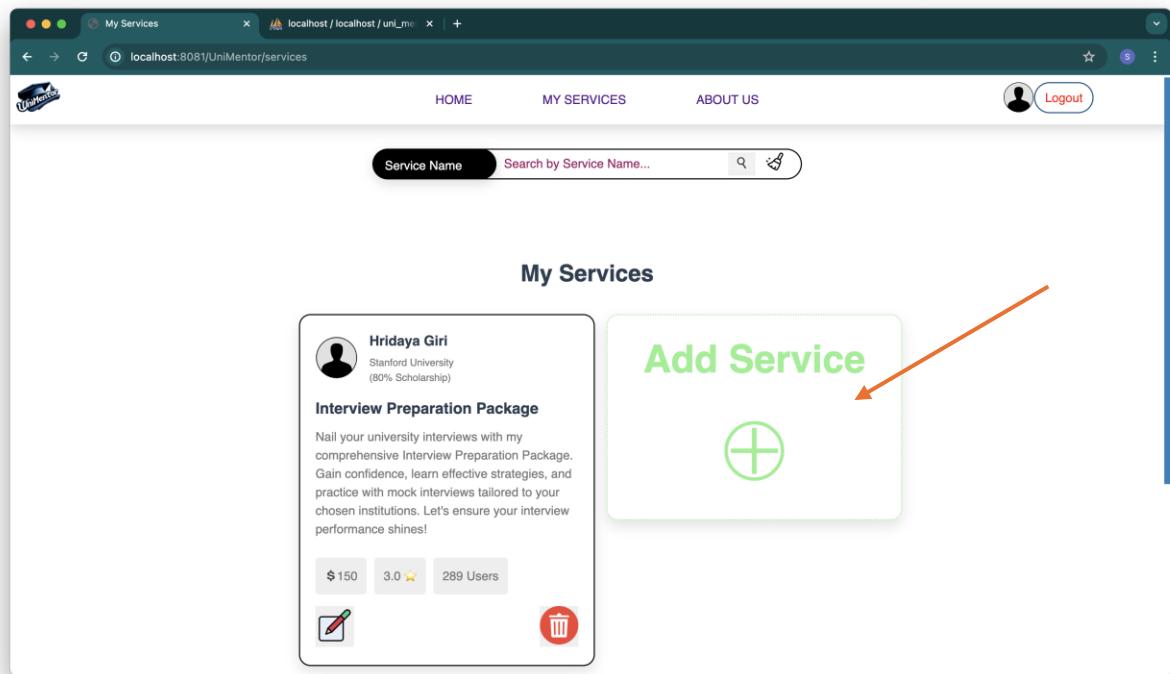


Figure 132: Screenshot of clicking add service button

2) Entering service details:

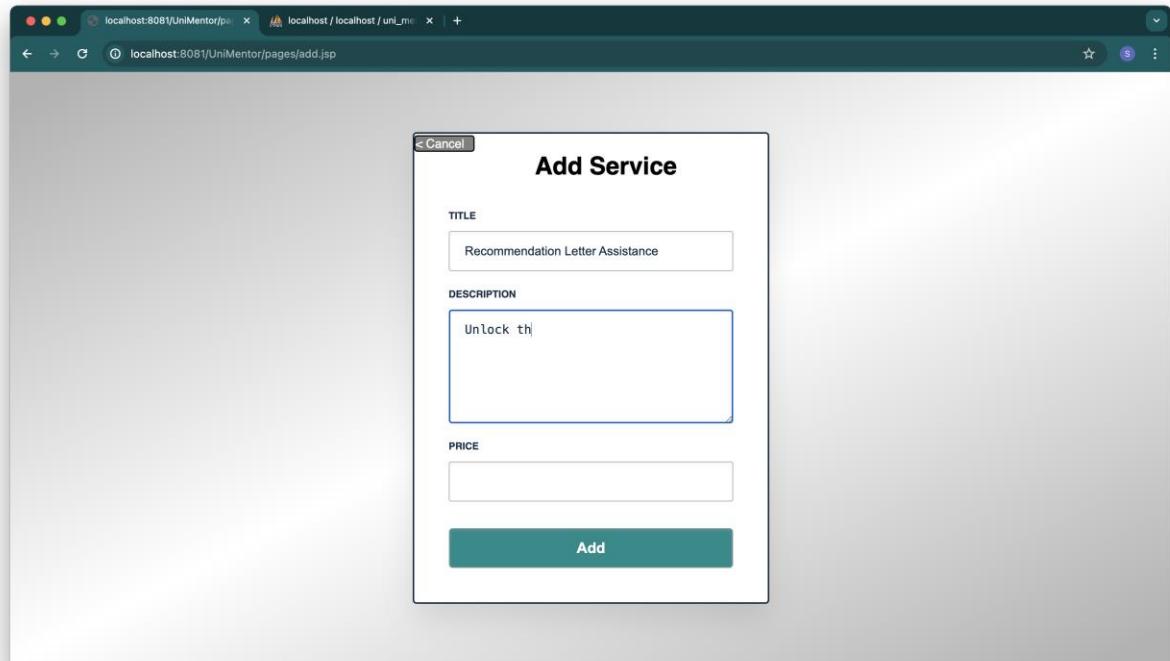


Figure 133: Screenshot of entering service details

3) Add button is clicked:

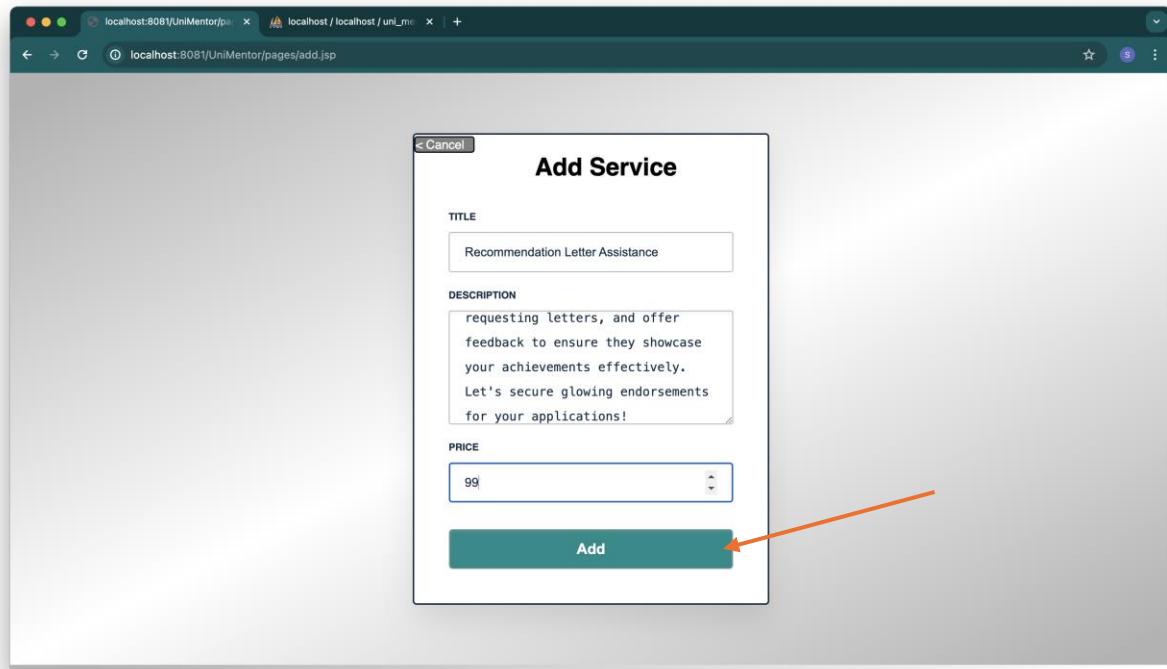


Figure 134: Screenshot of clicking add button

4) Added service displayed:

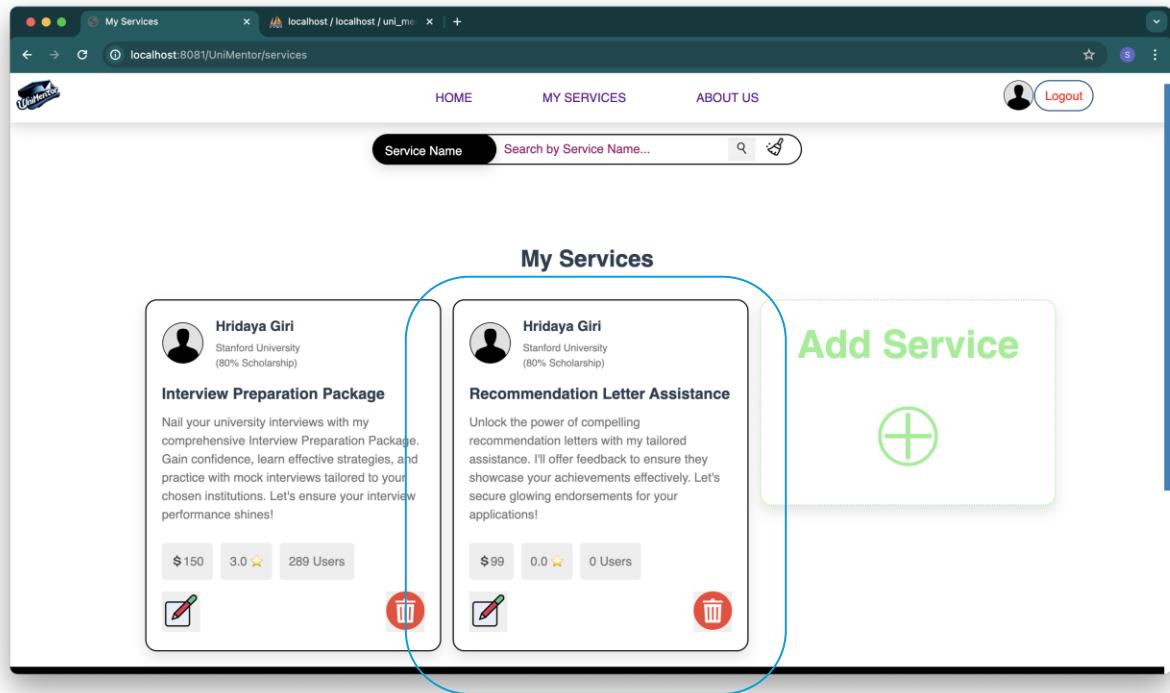


Figure 135: Screenshot of new service added

5) Added service displayed in database:

The screenshot shows the 'service' table in phpMyAdmin. The table has columns: service_id, service_title, description, price, creation_date, rating, service_users, status, and mentor_username. The last row, with service_id 75, is highlighted with a blue box. The row details are: service_title 'Recommendation Letter Assistance', description 'Unlock the power of compelling recommendation letter...', price '\$99', creation_date '2024-05-10', rating '0.0', service_users '0', status 'Active', and mentor_username 'Hridaya_12'.

	service_id	service_title	description	price	creation_date	rating	service_users	status	mentor_username
	72	Interview Preparation Package	Nail your university interviews with my comprehens...	150	2024-05-10	3.0	289	Active	Hridaya_12
	73	Full Application Process Guide	Get expert guidance for your university applicatio...	200	2024-05-10	4.3	19	Active	Sardul_11
	74	Writing Personal Statement	Struggling with your university personal statement...	100	2024-05-10	4.5	56	Active	Sardul_11
	75	Recommendation Letter Assistance	Unlock the power of compelling recommendation lett...	99	2024-05-10	0.0	0	Active	Hridaya_12

Figure 136: Screenshot of added service in database

Test 4.2. : Test for new added service displayed for seeker

Objective	To check the added service being displayed for seeker.
Action	<ul style="list-style-type: none"> ➤ Logged in as a seeker. ➤ View Services button clicked to navigate to all services.
Expected Result	The added service in test 4.1. should be displayed for seeker.
Actual Result	The added service was displayed.
Conclusion	The test is successful.

Table 8: Test table for added service displayed

Proofs:

- 1) Logging in as a seeker:

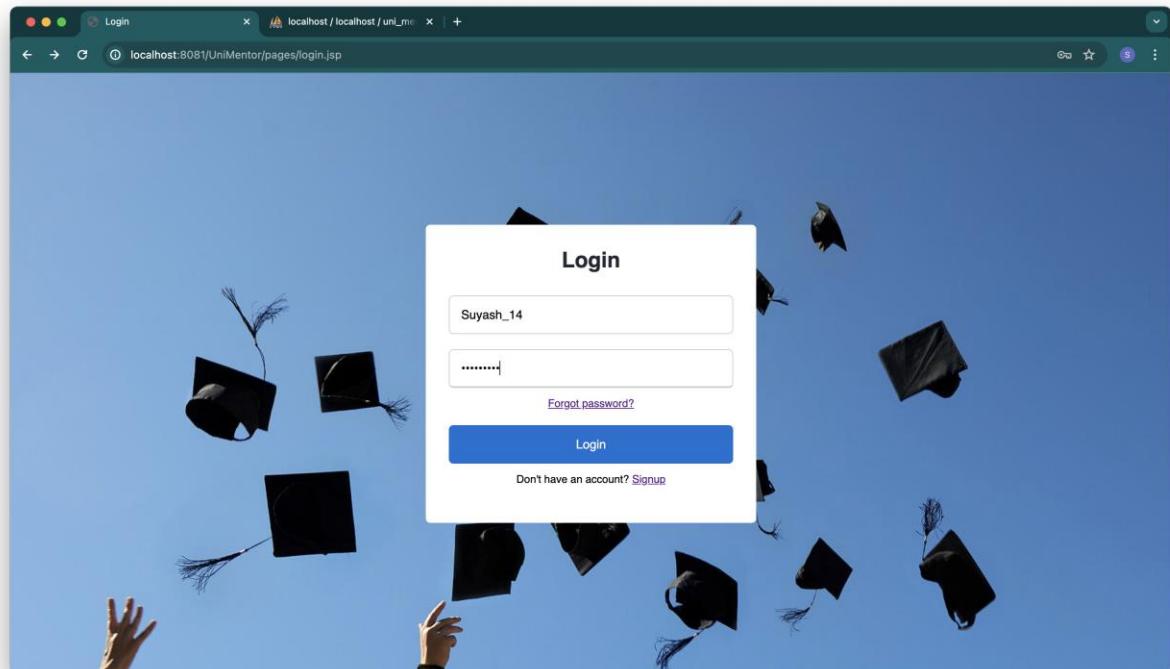


Figure 137: Screenshot of logging in as seeker

- 2) Clicking view services button to navigate to all services:

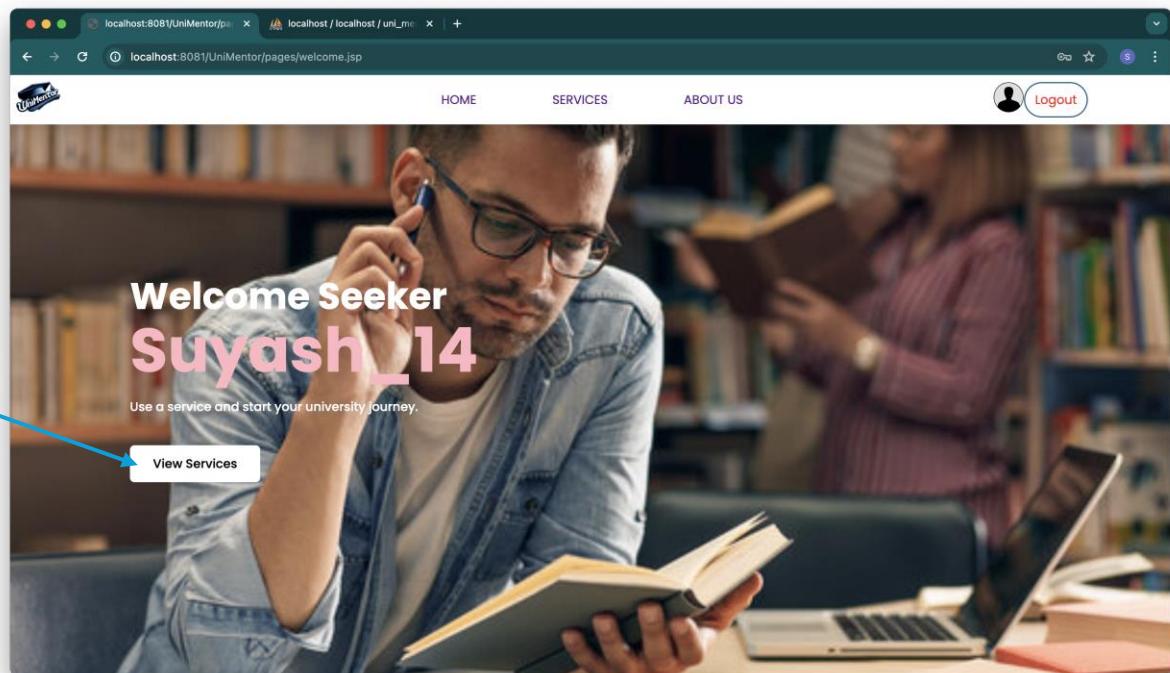


Figure 138: Screenshot of clicking view services button

- 3) The added service displayed:

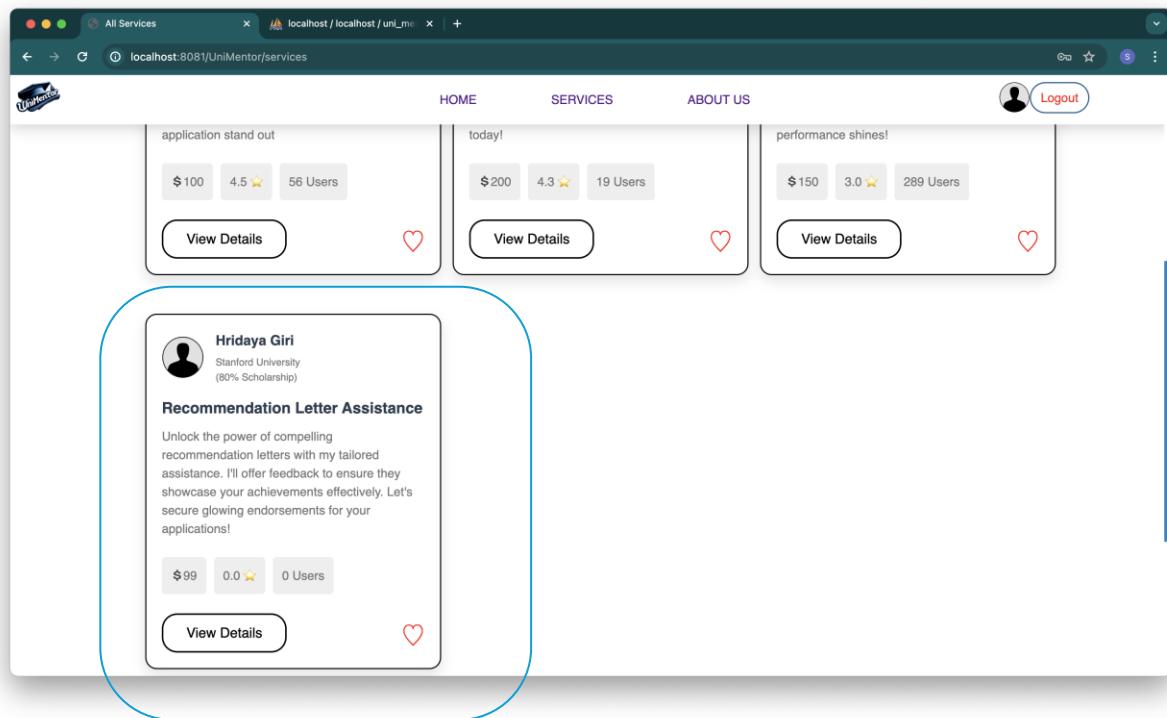


Figure 139: Screenshot of added service displayed

5.5. Test 5: Test for Update

Test 5.1. : Test for selected service being updated for mentor

Objective	To update the selected service.
Action	<ul style="list-style-type: none"> ➤ Update symbol button is clicked for a service in my services. ➤ Details of the selected service is updated. ➤ Update button is clicked.
Expected Result	The service should be updated as entered details.
Actual Result	The service was updated.
Conclusion	The test is successful.

Table 9: Test table for selected service update

Proofs:

1) Clicking update symbol button for a service in my services:

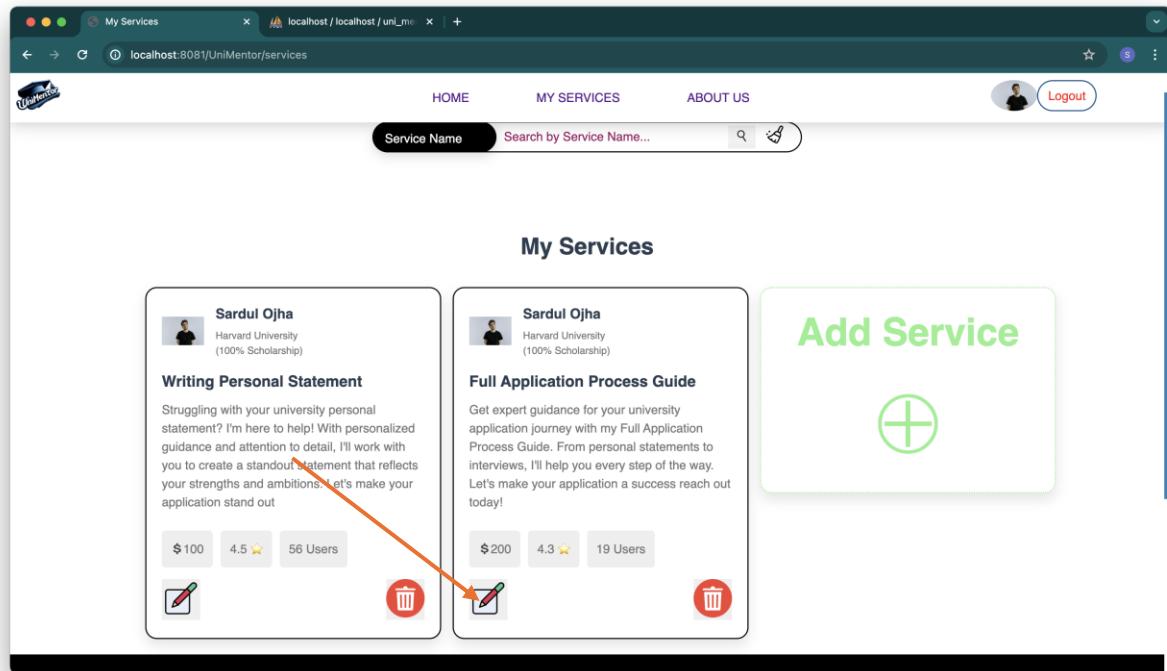


Figure 140: Screenshot of clicking update button

2) Updating details of the selected service:

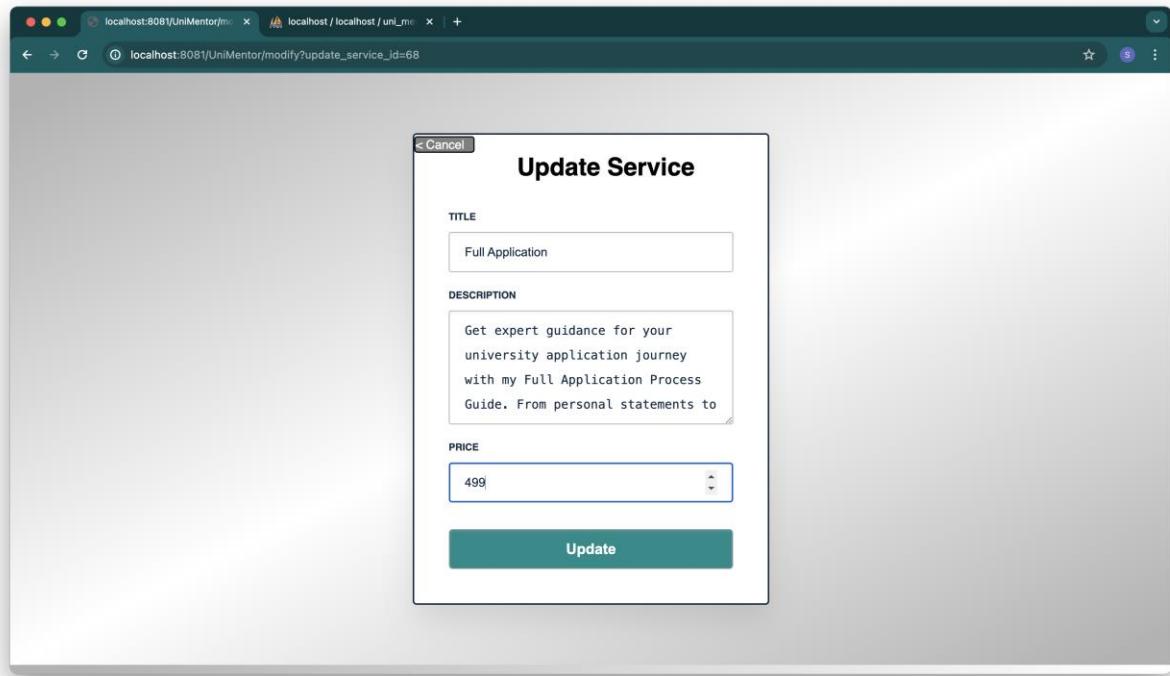


Figure 141: Screenshot of details updated

3) Clicking update button:

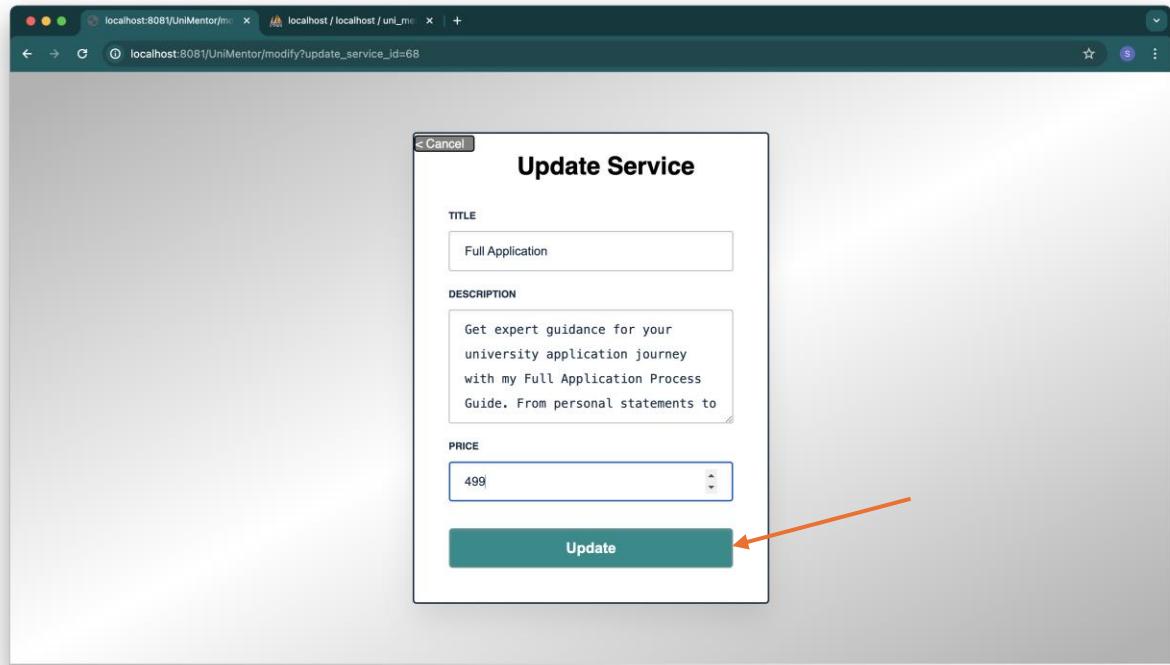


Figure 142: Screenshot of update button clicked

4) Change in service displayed:

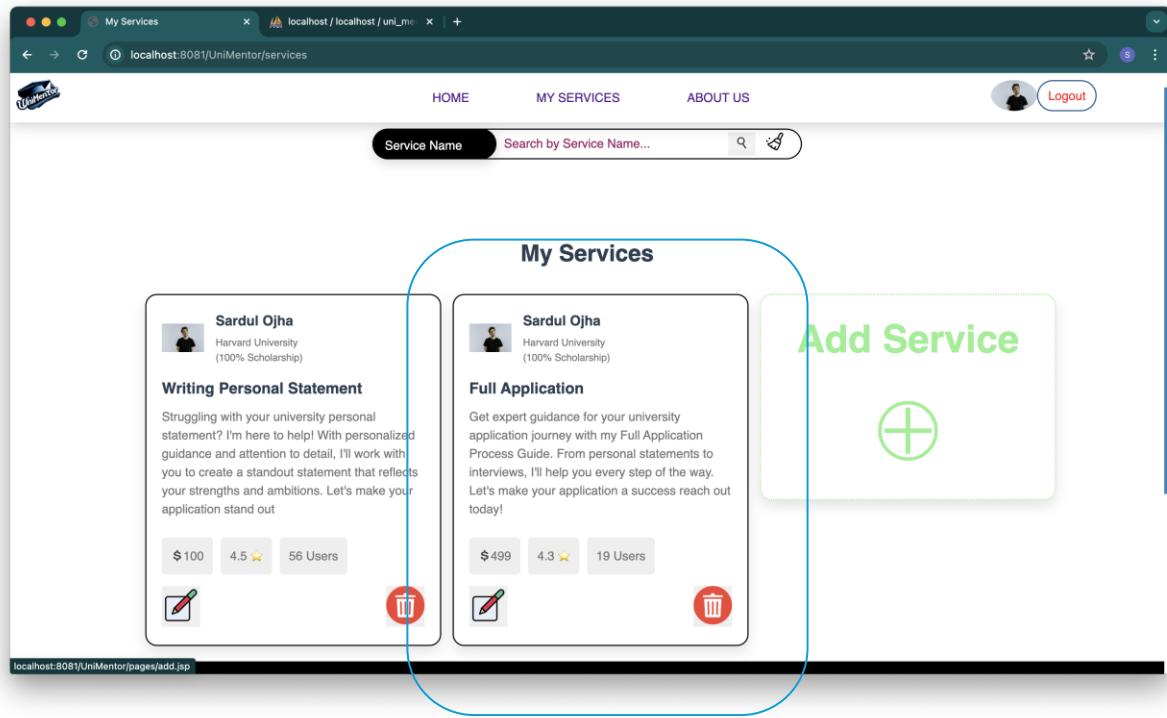


Figure 143: Screenshot of change in my service

5) Change in service displayed in database:

The screenshot shows the phpMyAdmin interface for a database named 'uni_mentor'. The left sidebar shows various databases and tables. The main area displays the 'service' table with the following data:

service_id	service_title	description	price	creation_date	rating	service_users	status	mentor_username
72	Interview Preparation Package	Nail your university interviews with my comprehensive...	150	2024-05-10	3.0	288	Active	Hridaya_12
73	Full Application	Get expert guidance for your university application...	499	2024-05-10	4.3	19	Active	Sardul_11
74	Writing Personal Statement	Struggling with your university personal statement...	100	2024-05-10	4.5	56	Active	Sardul_11
76	Recommendation Letter Assistance	Unlock the power of compelling recommendation lett...	99	2024-05-10	0.0	0	Active	Hridaya_12

Figure 144: Screenshot of change in database

Test 5.2. : Test for updated service displayed for seeker

Objective	To check the updated service being displayed for seeker.
Action	<ul style="list-style-type: none"> ➤ Logged in as a seeker. ➤ View Services button clicked to navigate to all services.
Expected Result	The updated service in test 5.1. should be displayed with updated details.
Actual Result	The updated service was displayed.
Conclusion	The test is successful.

Table 10: Test table for updated service display

Proofs:

- 1) Logging in as a seeker:

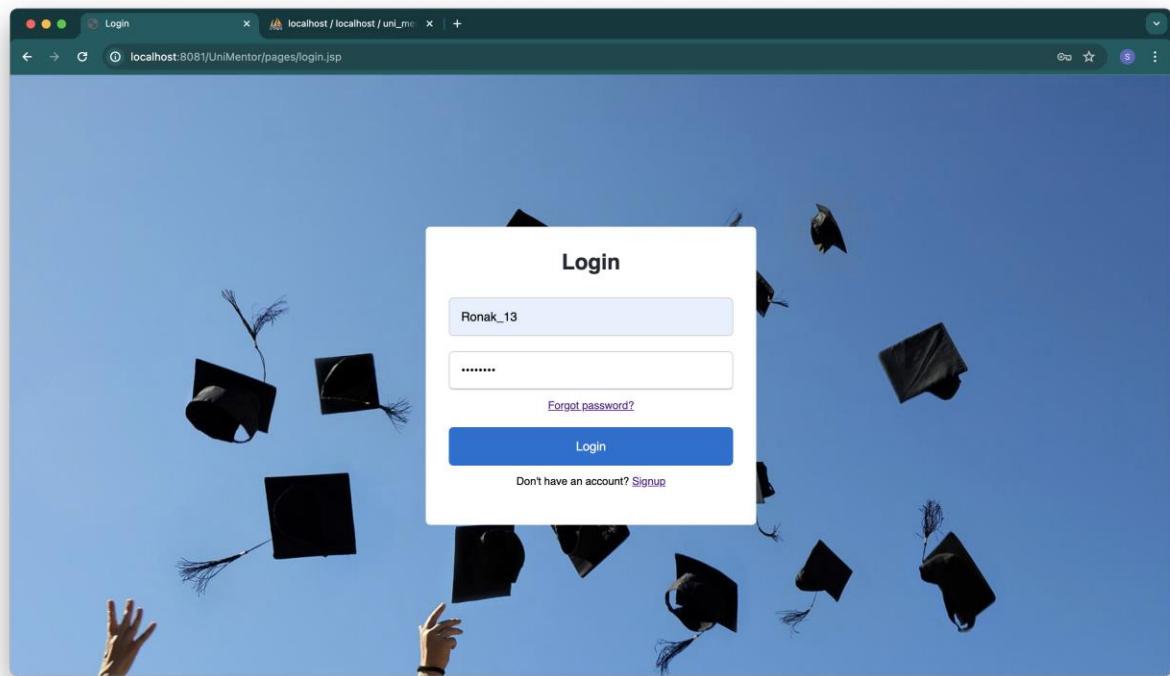


Figure 145: Screenshot of logging in as a seeker

2) Clicking view services button to navigate to all services:

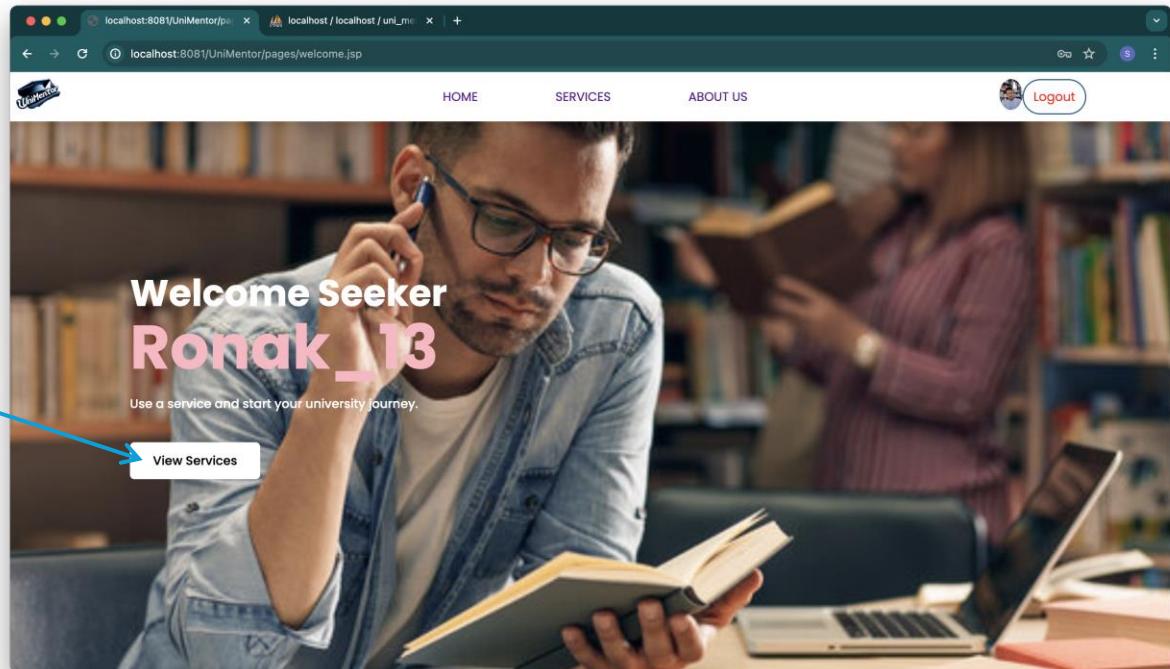


Figure 146: Screenshot of clicking view services

3) The updated service displayed:

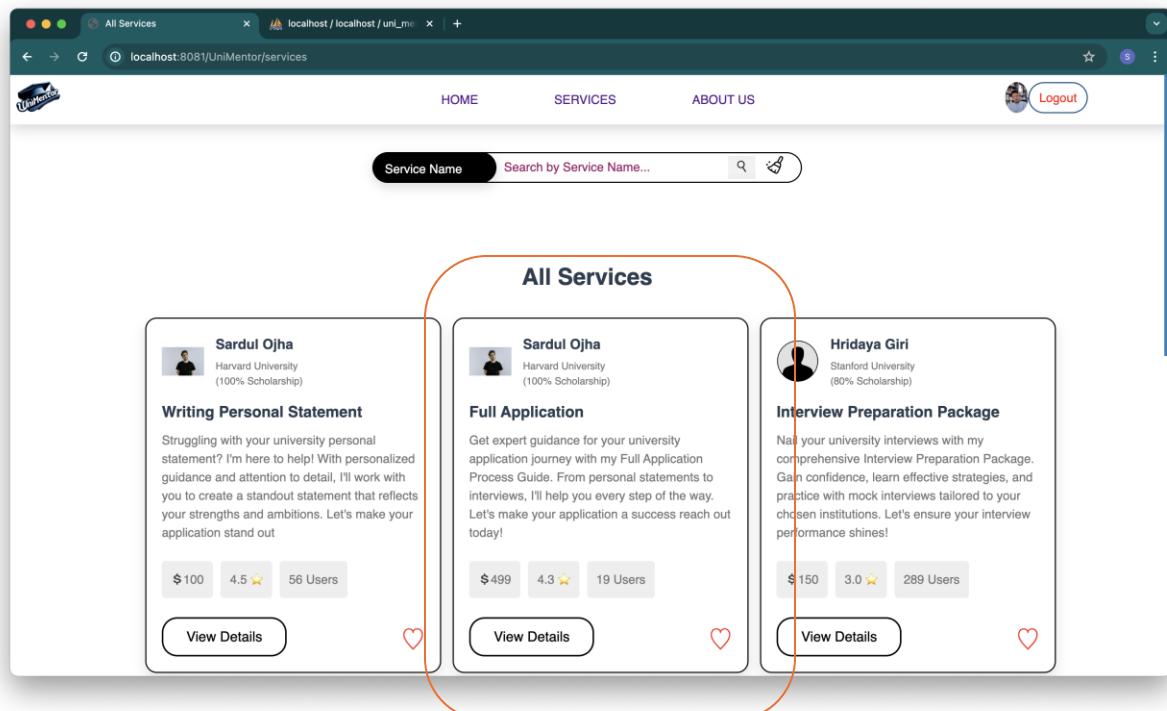


Figure 147: Screenshot of updated service displayed

5.6. Test 6: Test for Delete

Test 6.1. : Test for selected service being deleted for mentor

Objective	To delete the selected service.
Action	<ul style="list-style-type: none"> ➤ Delete button is clicked for a service in my services. ➤ Yes button is clicked for confirmation.
Expected Result	The service should be deleted and should not be displayed.
Actual Result	The deleted service was not displayed.
Conclusion	The test is successful.

Table 11: Test table for selected service delete

Proofs:

1) Clicking delete button for a service in my services:

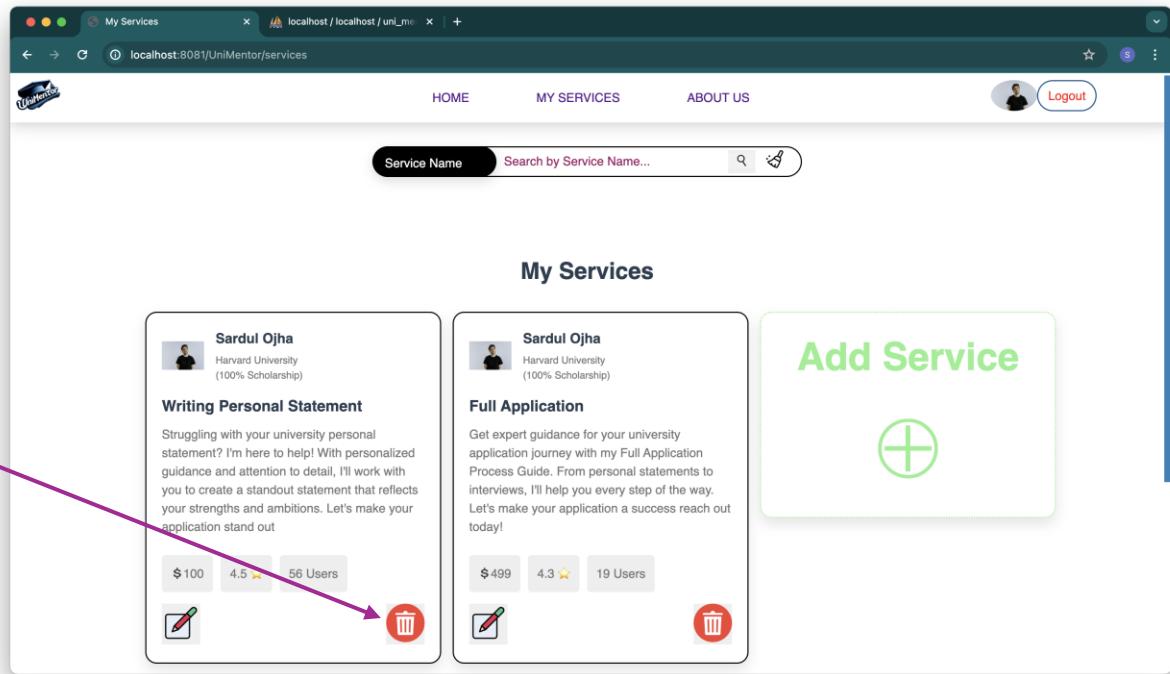


Figure 148: Screenshot of delete button clicked

2) Clicking yes button for confirmation:

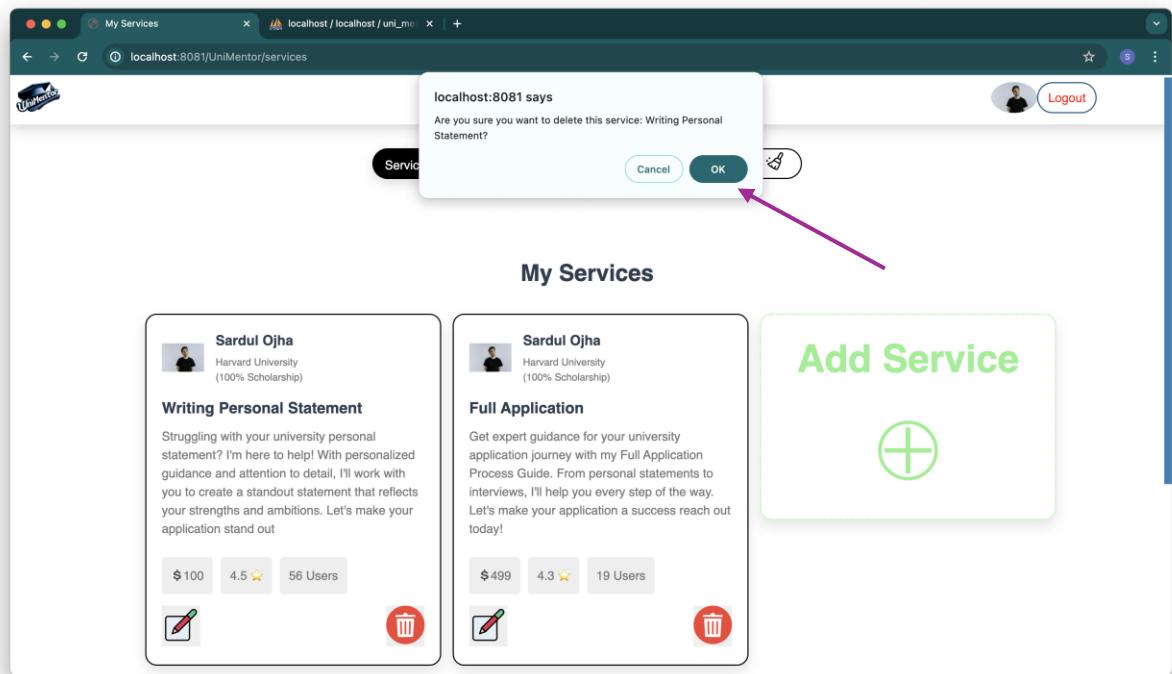


Figure 149: Screenshot of ok button clicked

3) The deleted service not displayed:

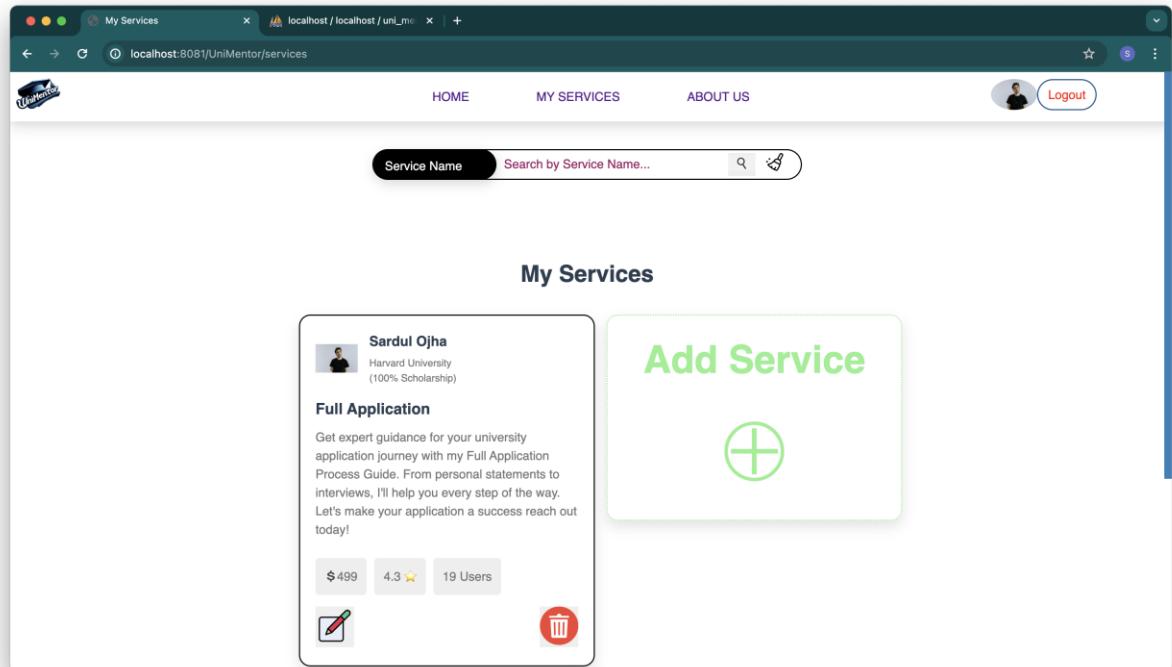
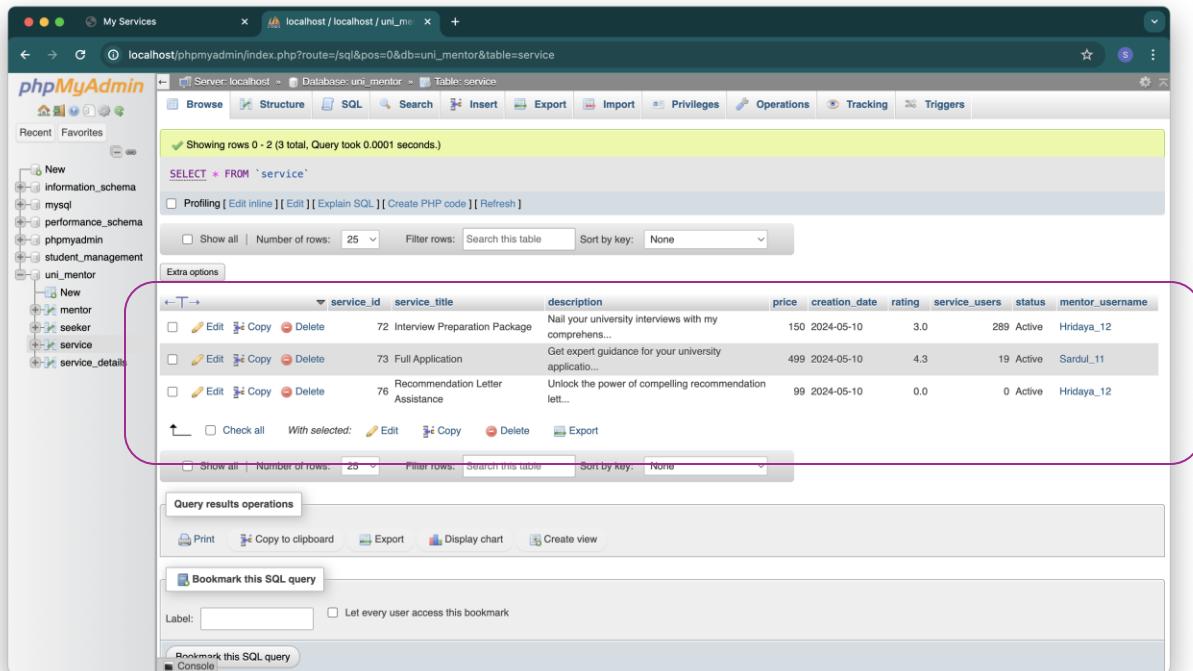


Figure 150: Screenshot of deleted service not displayed

4) The deleted service removed from database:



The screenshot shows the phpMyAdmin interface for the 'uni_mentor' database. The 'service' table is selected. A red box highlights the table data area, showing three rows that have been deleted. The columns are labeled: service_id, service_title, description, price, creation_date, rating, service_users, status, and mentor_username. The rows correspond to service IDs 72, 73, and 76.

service_id	service_title	description	price	creation_date	rating	service_users	status	mentor_username
72	Interview Preparation Package	Nail your university interviews with my comprehens...	150	2024-05-10	3.0	289	Active	Hridaya_12
73	Full Application	Get expert guidance for your university applicatio...	499	2024-05-10	4.3	19	Active	Sardul_11
76	Recommendation Letter Assistance	Unlock the power of compelling recommendation lett...	99	2024-05-10	0.0	0	Active	Hridaya_12

Figure 151: Screenshot of deleted service removed from database

Test 6.2. : Test for deleted service not displayed for seeker

Objective	To check the deleted service not being displayed for seeker.
Action	<ul style="list-style-type: none"> ➤ Logged in as a seeker. ➤ All Services button clicked to navigate to all services.
Expected Result	The deleted service in test 6.1. should not be displayed.
Actual Result	The deleted service was not displayed.
Conclusion	The test is successful.

Table 12: Test table for deleted service not displayed for seeker

Proofs:

- 1) Logging in as a seeker:

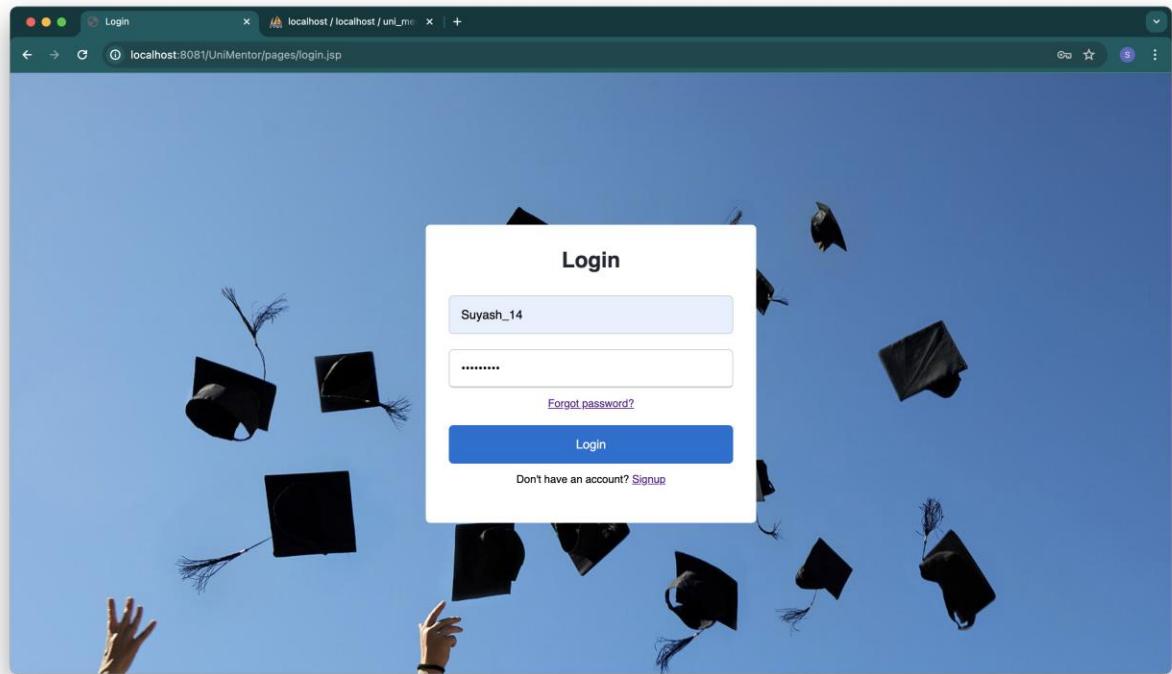
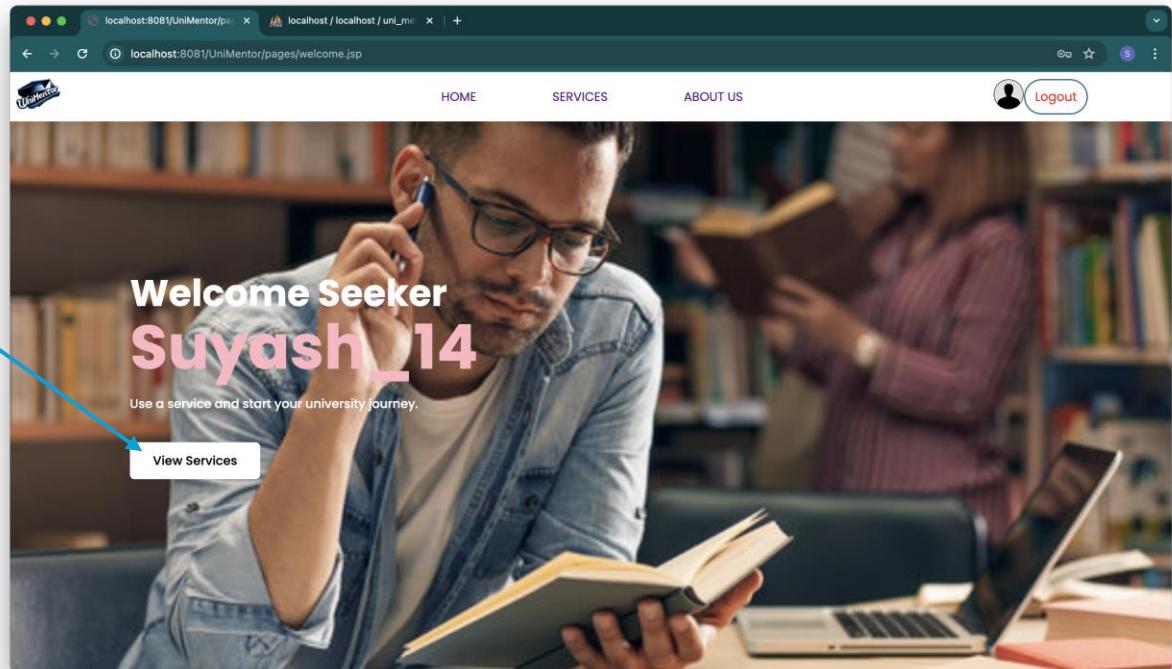


Figure 152: Screenshot of logging in as seeker

2) Clicking view services button to navigate to all services:



3) The deleted service not displayed:

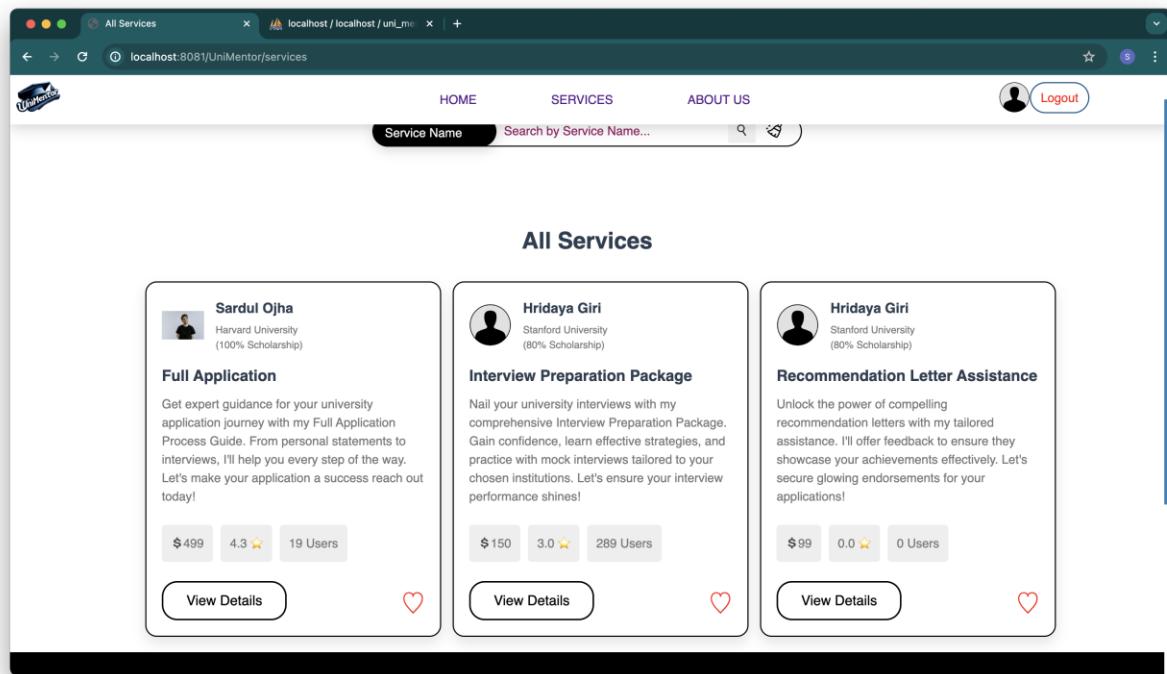


Figure 153: Screenshot of deleted service not displayed for seeker

5.7. Test 7: Test for Profile (C.R.U.D)

Test 7.1. : Test for new profile being displayed (Create and Read)

Objective	To check the information being displayed in profile after logging in.
Action	<ul style="list-style-type: none"> ➤ Logged in as user (mentor or seeker). ➤ Profile button is clicked on the navbar.
Expected Result	All the details of the logged in user should be displayed.
Actual Result	All the details was displayed.
Conclusion	The test is successful.

Table 13: Test table for profile display

Proofs:

1) Logging in as user:

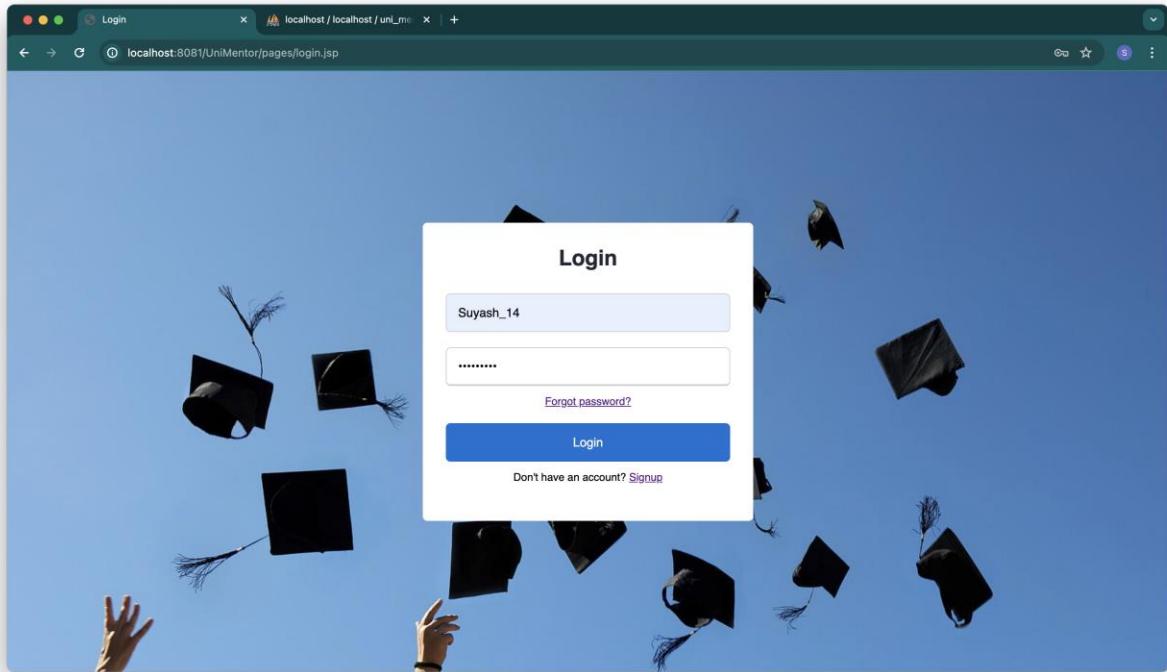


Figure 154: Screenshot of logging in as a user

2) Clicking profile button on the navbar:

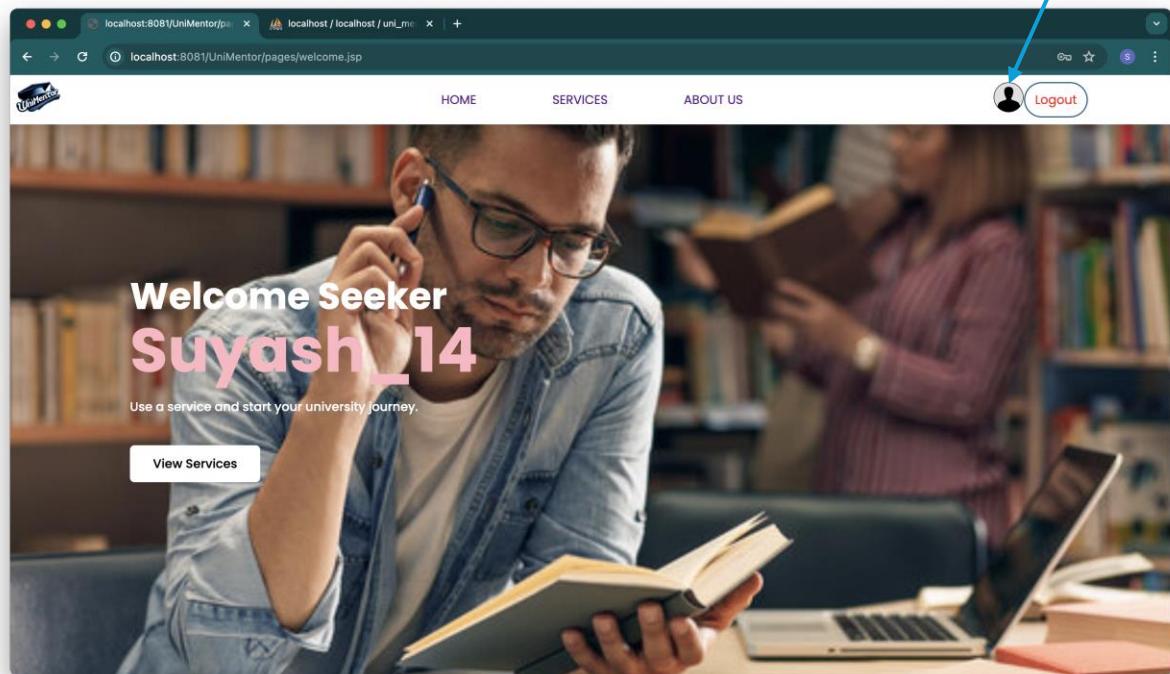


Figure 155: Screenshot of clicking profile button

3) Profile information displayed:

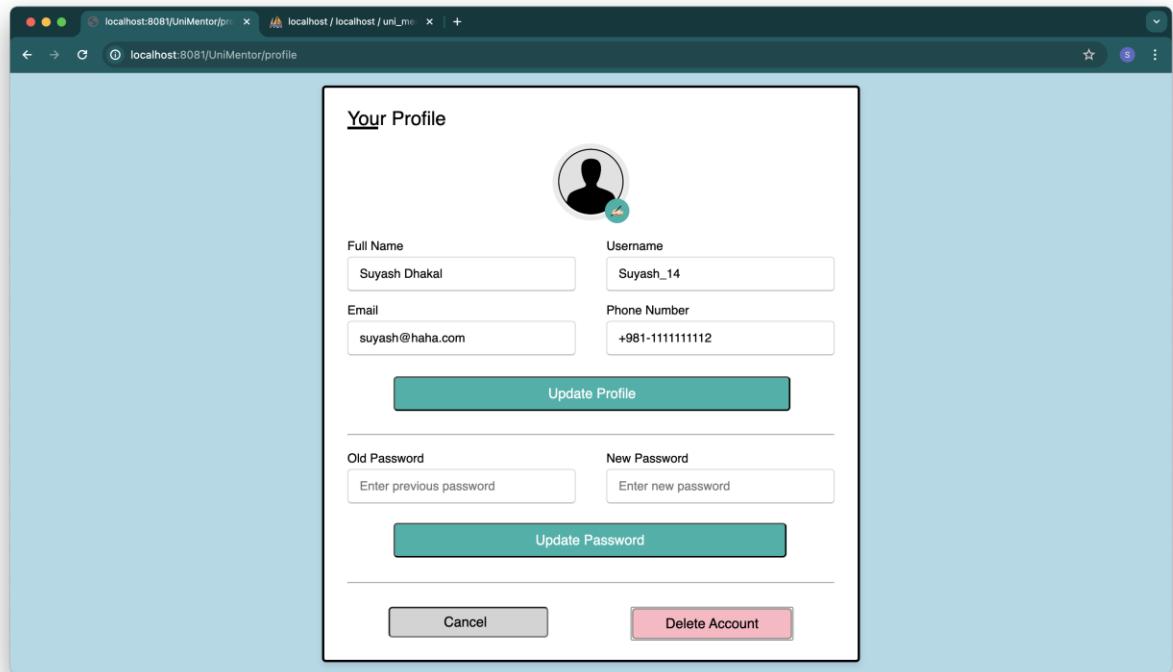


Figure 156: Screenshot of profile information displayed

4) Profile information in database:

	seeker_username	seeker_name	seeker_phone	seeker_email	seeker_password	seeker_photo	seeker_location
<input type="checkbox"/>	Edit Copy Delete Ness_16	Ness Shrestha	+981-111118890	ness@gmail.com	i3nfeWstXDr6/Ze2ZKwPD3FOMaMCr3h5gMrkAYsxGhtxVuFy...	img-1.jpg	Bhaktapur
<input checked="" type="checkbox"/>	Edit Copy Delete Renek_10	Renek Shrestha	+977-888888831	renek@yahoo.com	ANWR6Te8FMqCwSh0VQ21DlXqzZ_G1WhgClf77QF04D7gs1VU...	img-4.jpg	Kathmandu, Nepal
<input type="checkbox"/>	Edit Copy Delete Suyash_14	Suyash Dhakal	+981-111111112	suyash@haha.com	agTdBi32HvwirZY44UxRT8hpDNx0c/qoyz3Fc4ejpnAkdy/FT...	default-profile.png	Beijing, China

Figure 157: Screenshot of profile information in database

Test 7.2. : Test for profile being updated (Update)

Objective	To update the profile information.
Action	<ul style="list-style-type: none"> ➤ Profile page is opened. ➤ Personal details are updated. ➤ Update Profile button is clicked. ➤ Profile page is again opened to check results.
Expected Result	<ul style="list-style-type: none"> ➤ The profile page should be updated according to the updated information.
Actual Result	The profile page was updated.
Conclusion	The test is successful.

Table 14: Test table for profile update

Proofs:

- 1) Clicking profile button on the navbar:

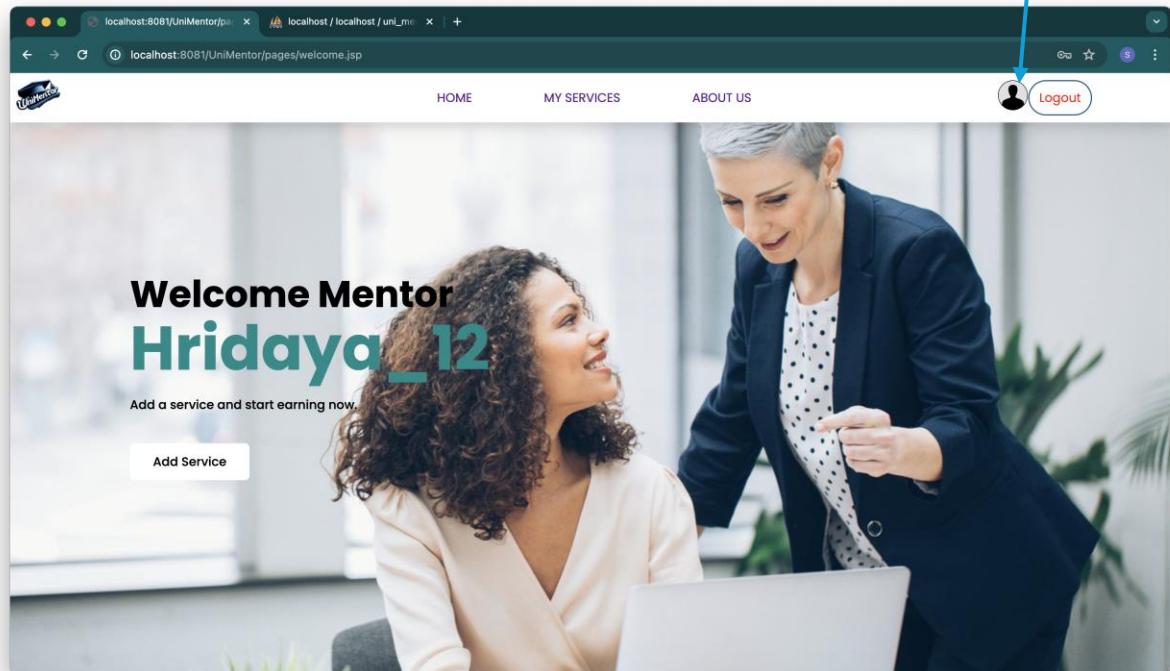


Figure 158: Screenshot of clicking profile button on navbar

2) Profile information displayed:

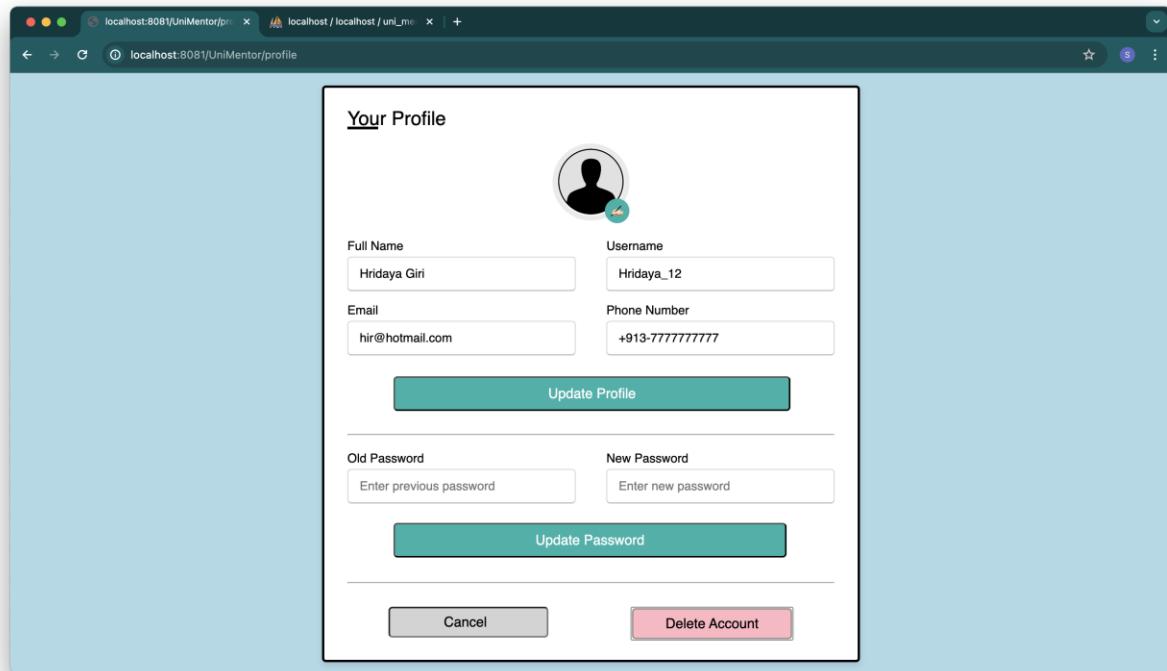


Figure 159: Screenshot of profile information displayed

3) Updating profile information:

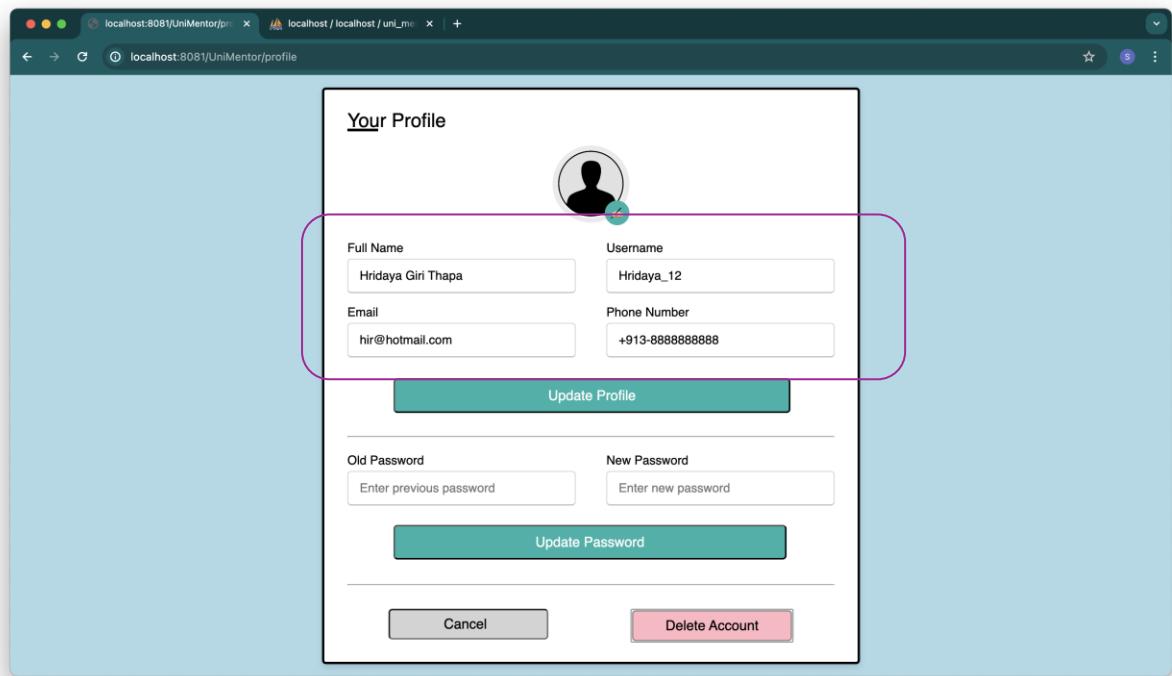


Figure 160: Screenshot of profile information updated

4) Clicking update profile button:

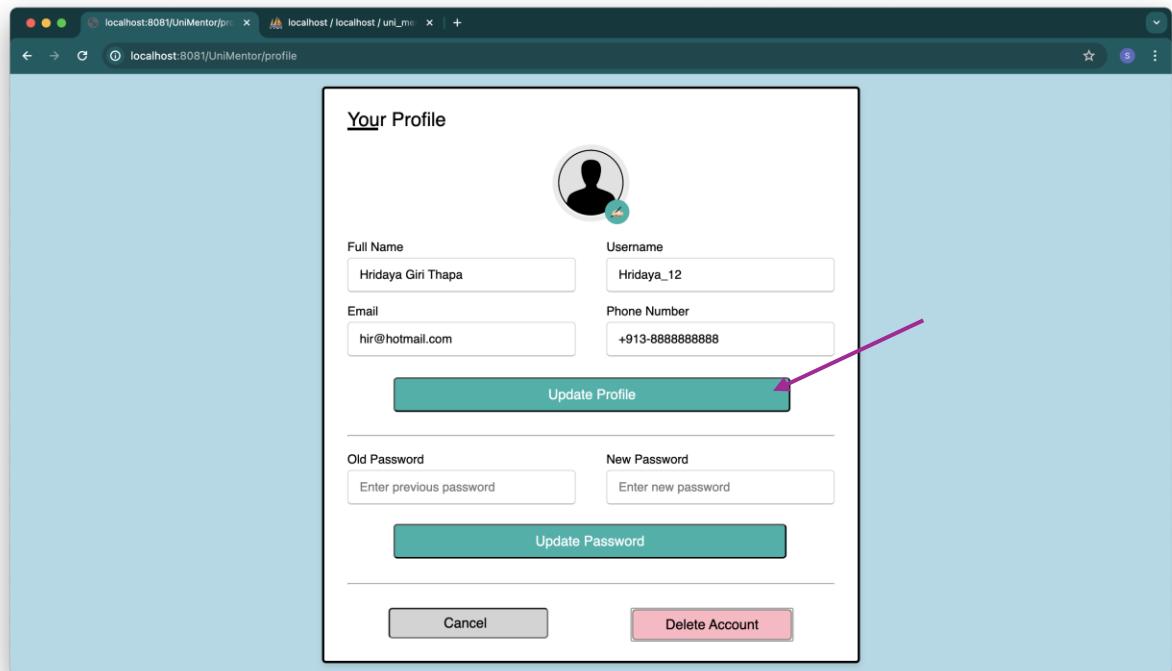


Figure 161: Screenshot of update profile button clicked

- 5) Again, clicking profile button on the navbar:

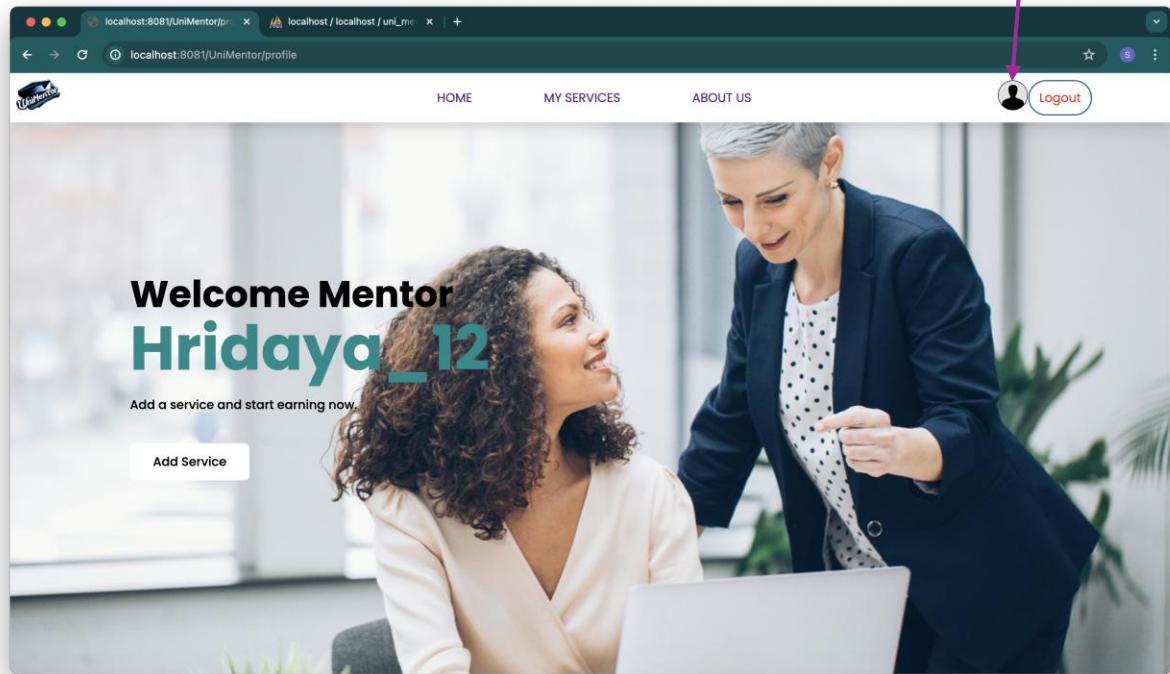


Figure 162: Screenshot of again clicking on profile button

- 6) Updated profile information:

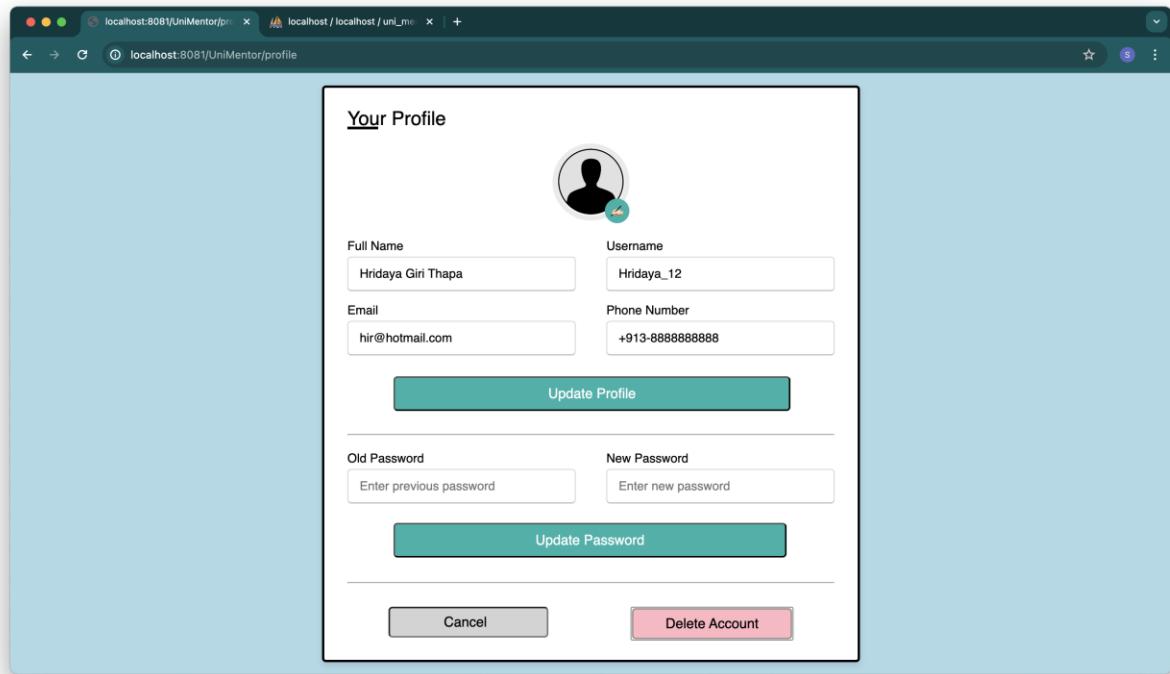


Figure 163: Screenshot of updated information displayed

7) Updated profile information in database:

	mentor_username	mentor_name	mentor_phone	mentor_email	mentor_password	mentor_photo	university_name
<input type="checkbox"/>	Edit	Sek_Md_15	+913-777712345	sek@gmail.com	GswrmN5zDMIOh7PrZOTx4BpnunAh7ySgug9UkFDHoTbAFrX4wn...	img-3.jpg	MIT
<input checked="" type="checkbox"/>	Edit	Sardul_11	+977-972855801	sardul@gmail.com	NPGKK7AE51Jlpn4M4h1YgRob84qJ4uHu01MMbwM2yQ3zm3uNzKW...	img-2.jpg	Harvard Univers
<input type="checkbox"/>	Edit	Hridaya_12	+913-888888888	hir@hotmail.com	Svdnj836W7Y1jTDx1YmOvLBj7D8T4pi++eDMzGsTVJvSkS84...	default-profile.png	Stanford Univers

Figure 164: Screenshot of information updated in database

Test 7.3. : Test for profile being deleted

Objective	To delete the account(profile).
Action	<ul style="list-style-type: none"> ➤ Logging in as a user (mentor or seeker). ➤ Profile page is opened. ➤ Delete profile button is clicked. ➤ Yes button is clicked for confirmation. ➤ Log in information is entered to check the account.
Expected Result	The account (profile) should be deleted.
Actual Result	The account was deleted.
Conclusion	The test is successful.

Table 15: Test table for profile delete

Proofs:

- 1) Clicking profile button on the navbar:

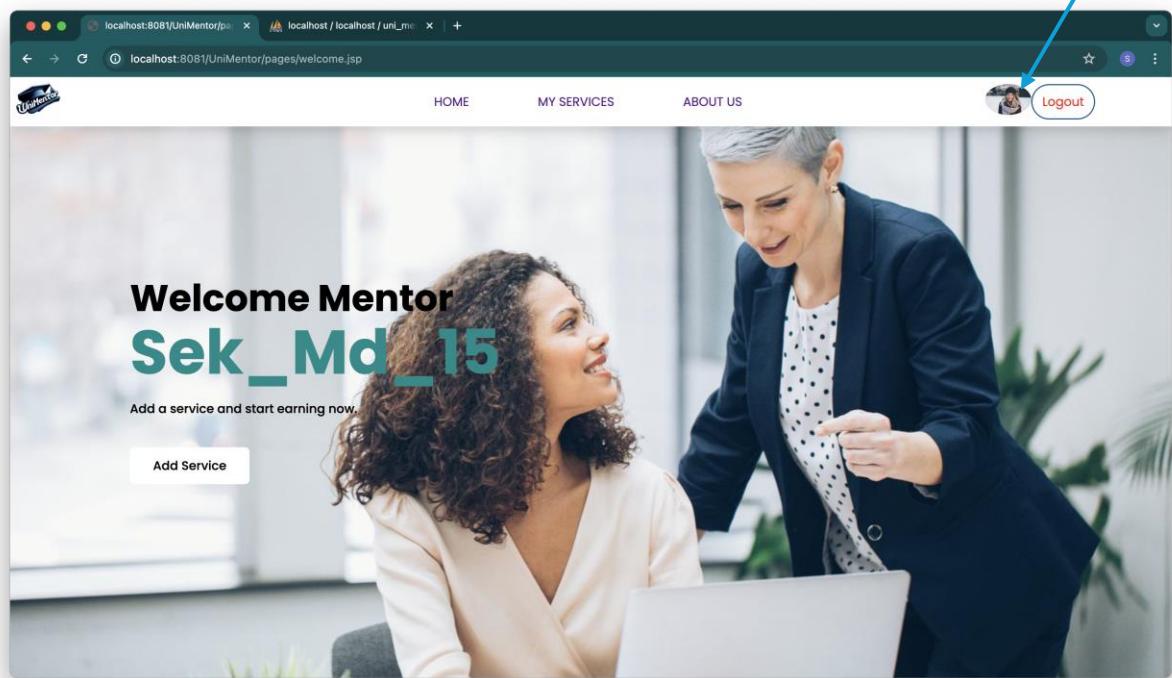


Figure 165: Screenshot of clicking profile buttons

2) Profile information displayed:

A screenshot of a web browser displaying the 'profile' page of the UniMentor application. The page is titled 'Your Profile' and contains a form for updating personal information. It includes fields for 'Full Name' (Sek Md Abid), 'Username' (Sek_Md_15), 'Email' (sek@gmail.com), and 'Phone Number' (+913-7777712345). There are two main sections for password changes: 'Update Profile' and 'Update Password'. Each section has 'Old Password' and 'New Password' fields. At the bottom of the page are 'Cancel' and 'Delete Account' buttons.

Figure 166: Screenshot of profile information displayed

3) Clicking delete profile button:

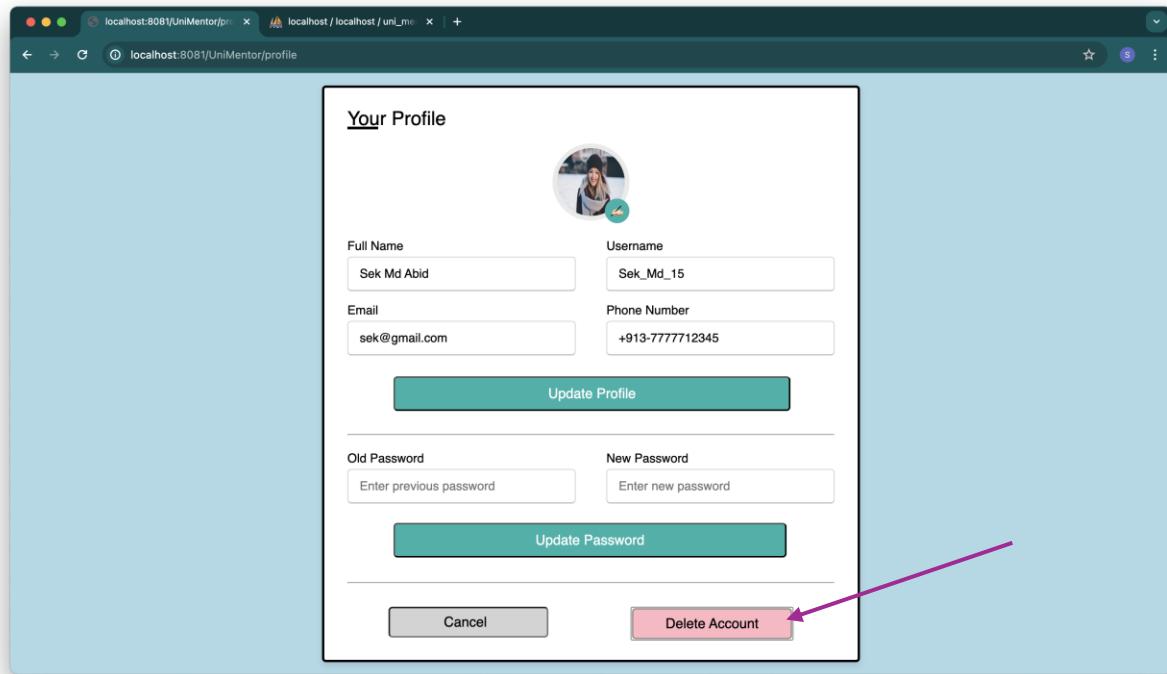


Figure 167: Screenshot of clicking delete button

4) Clicking yes for confirmation:

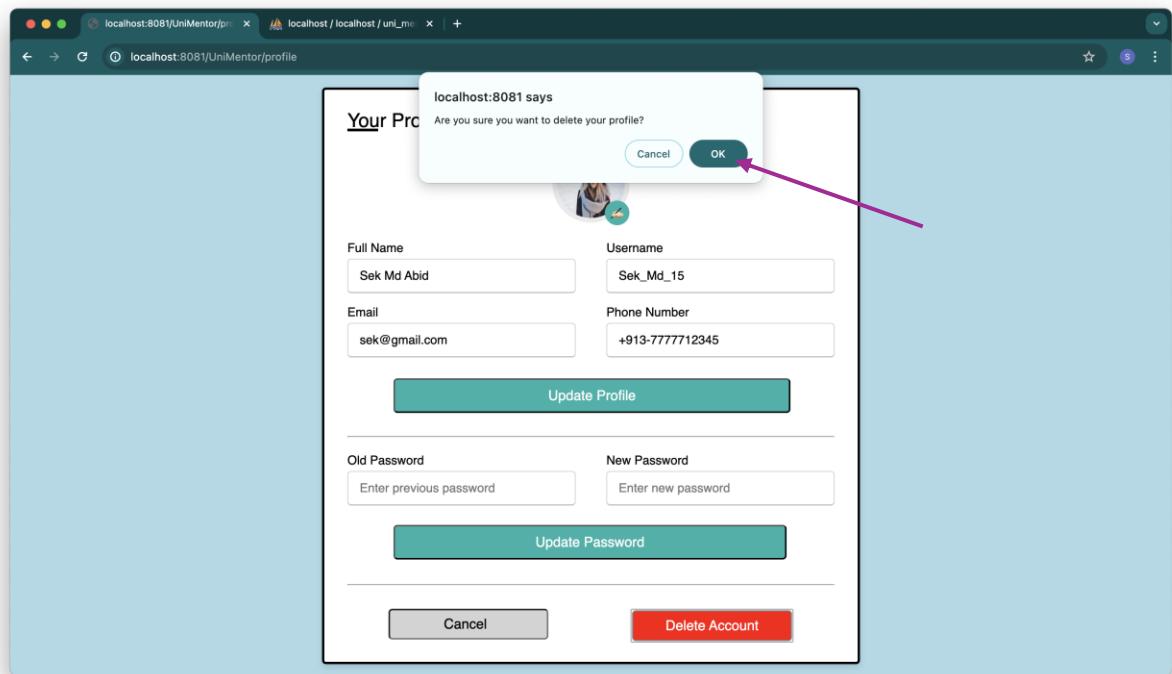


Figure 168: Screenshot of clicking yes for confirmation

5) Redirected to index page (logged out):

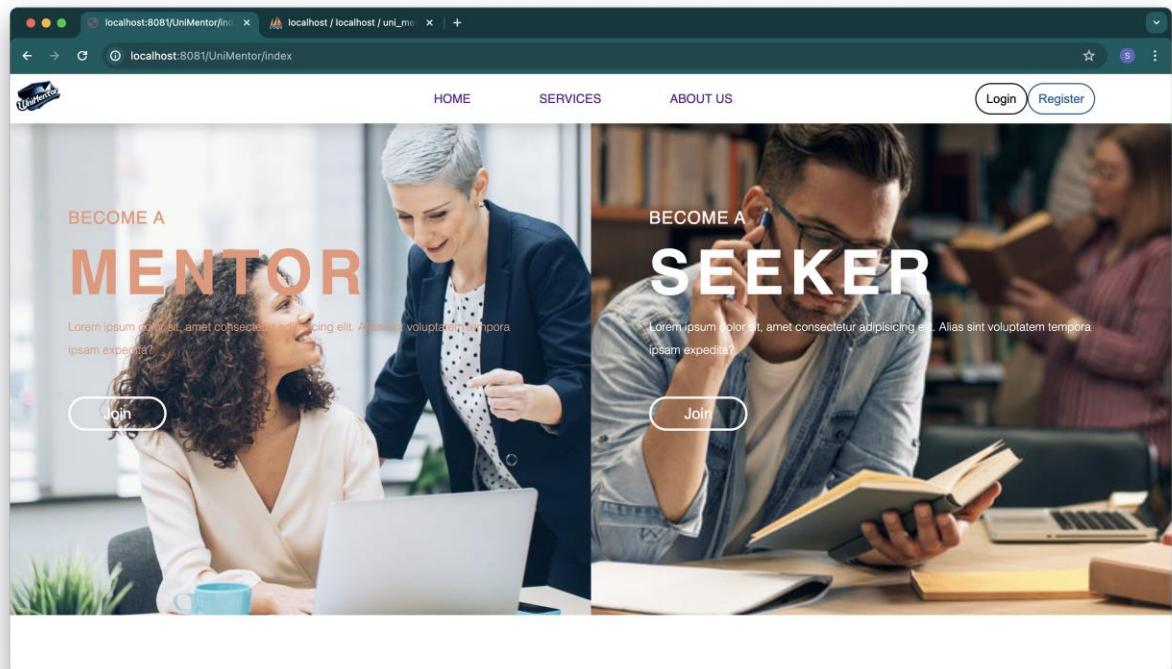
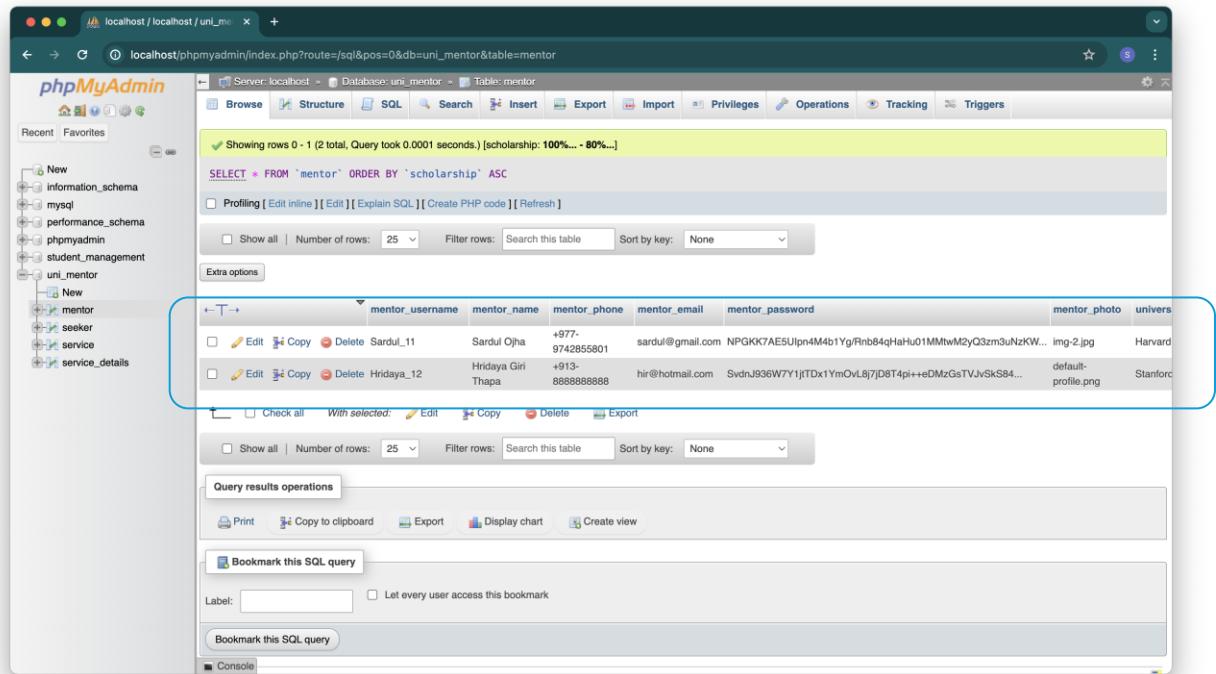


Figure 169: Screenshot of redirection to index page logged out

6) No account in database:



The screenshot shows the phpMyAdmin interface for the 'uni_mentor' database. The 'mentor' table is selected. The table data is as follows:

mentor_username	mentor_name	mentor_phone	mentor_email	mentor_password	mentor_photo	univers
Sardul_11	Sardul Ojha	+977-9742855801	sardul@gmail.com	NPGKK7AE5Ulpn4M4b1Yg/Rnb84qHaHu01MMtwM2yQ3zm3uNzK... img-2.jpg		Harvard
Hridaya_12	Hridaya Giri Thapa	+91-9888888888	hir@hotmail.com	SvdnjJ936W7Y1jITDx1YmOvL8j7D8T4pi++eDMzGsTVJvSkS84... default-profile.png		Stanford

Figure 170: Screenshot of account removed from database

7) Entering log in information to check:

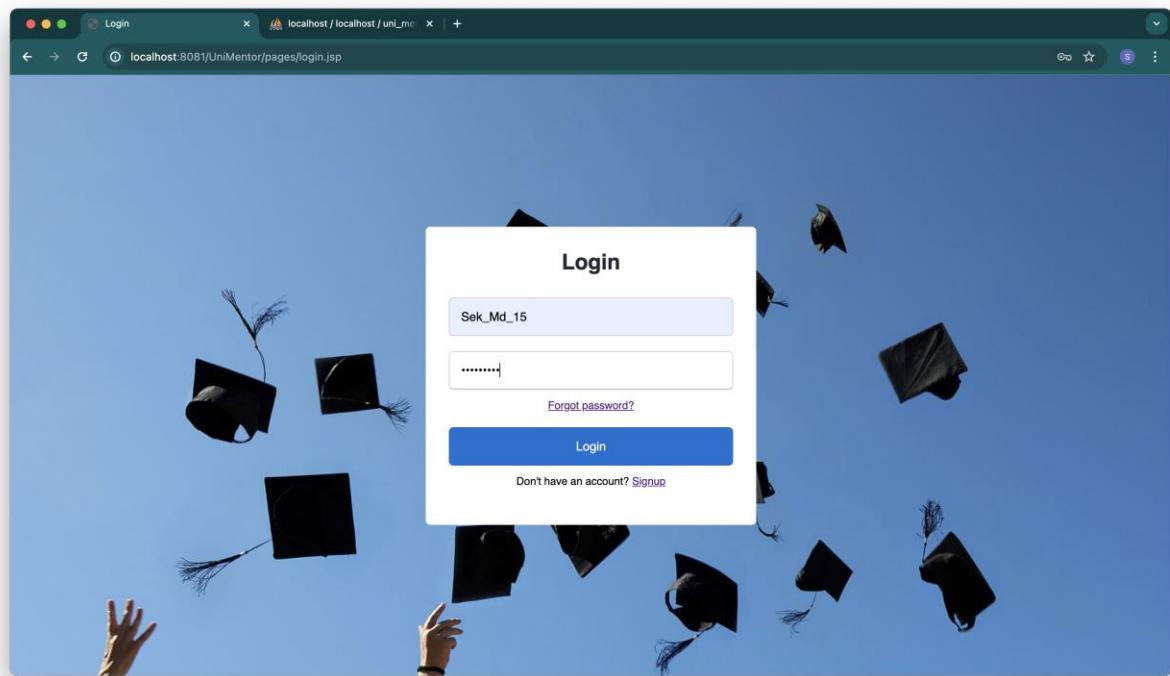


Figure 171: Screenshot of entering login information of deleted account

8) No account error message displayed:

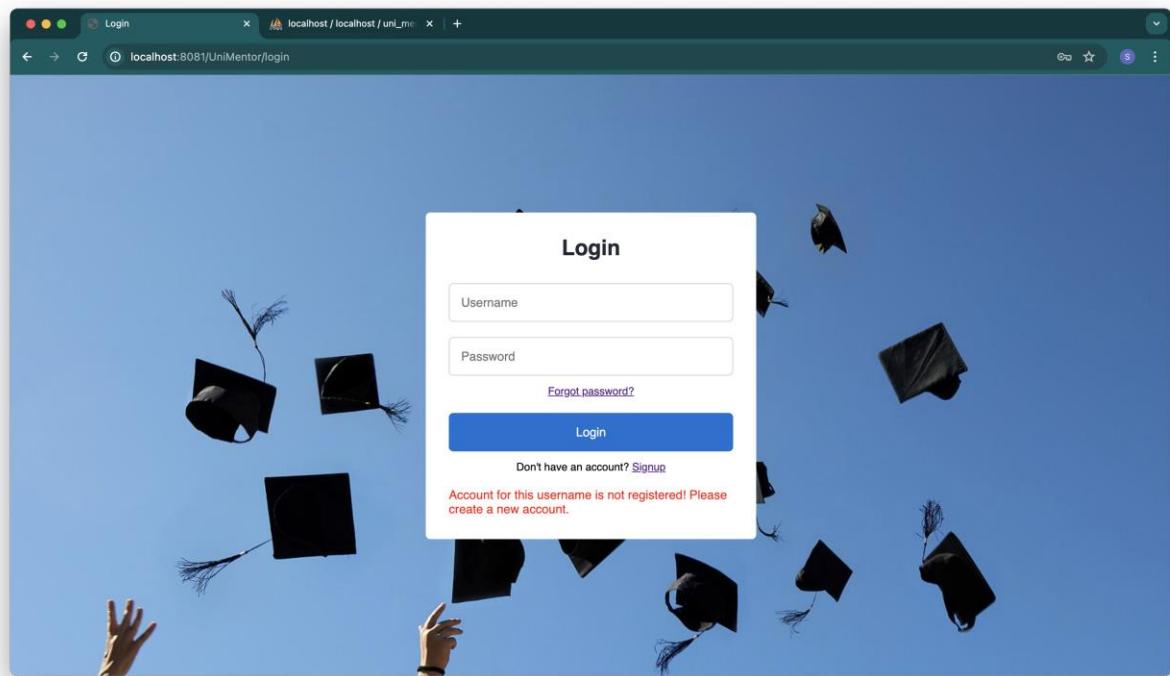


Figure 172: Screenshot of error message displayed

5.8. Test 8: Test for Password Reset

Test 8.1. : Test for input validation for password reset

Objective	To display error message when invalid details are entered for password reset.
Action	<ul style="list-style-type: none"> ➤ Profile page is clicked. ➤ Incorrect input is entered in password reset fields. ➤ Update password button is clicked.
Expected Result	Error message should be displayed indicating the error.
Actual Result	Error message was displayed.
Conclusion	The test is successful.

Table 16: Test table for validation for password reset

Proofs:

- 1) Clicking profile button on the navbar:

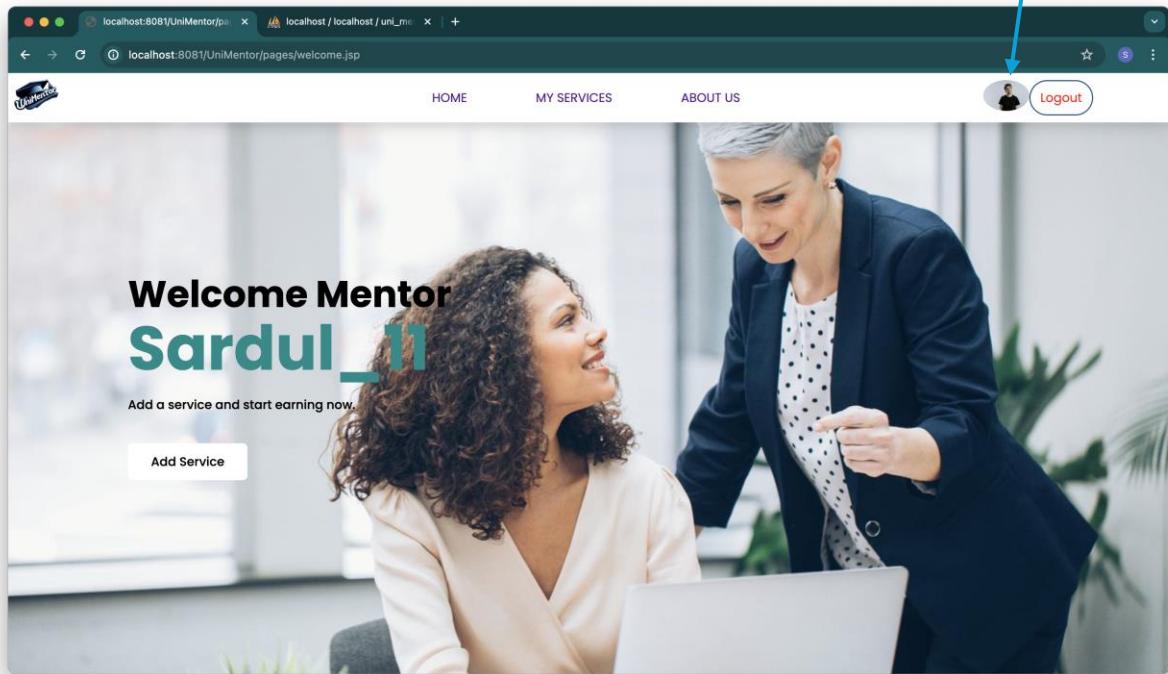


Figure 173: Screenshot of clicking profile button

- 2) Profile information displayed:

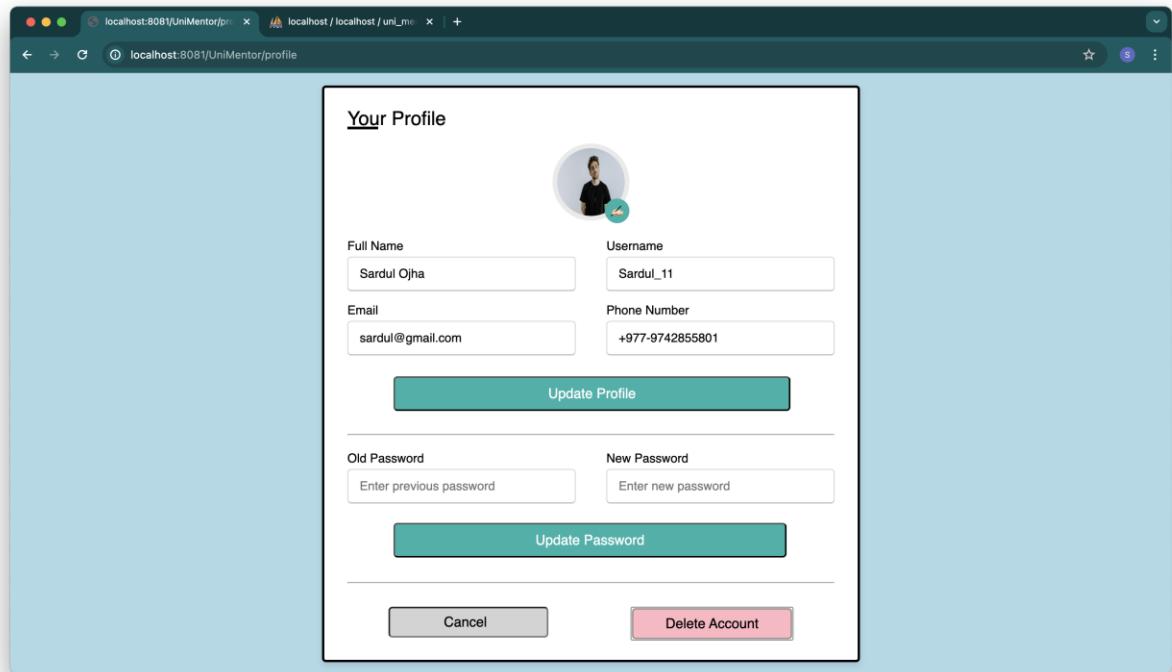


Figure 174: Screenshot of profile displayed

3) Entering invalid input in update password fields:

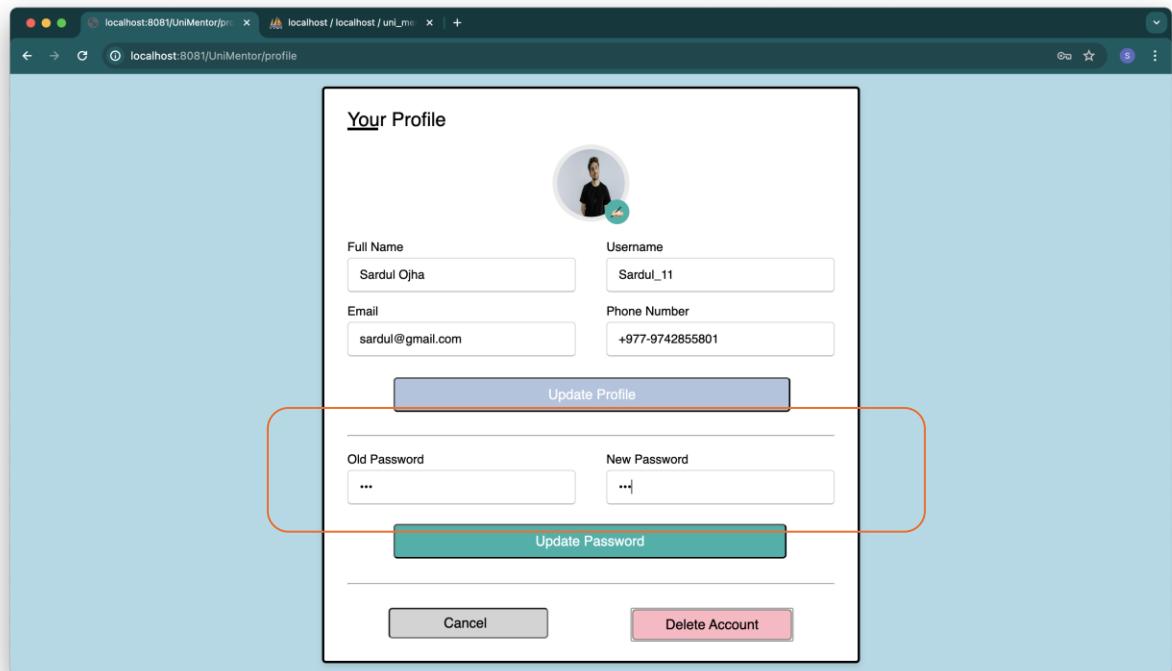


Figure 175: Screenshot of entering invalid in password reset

4) Clicking update password button:

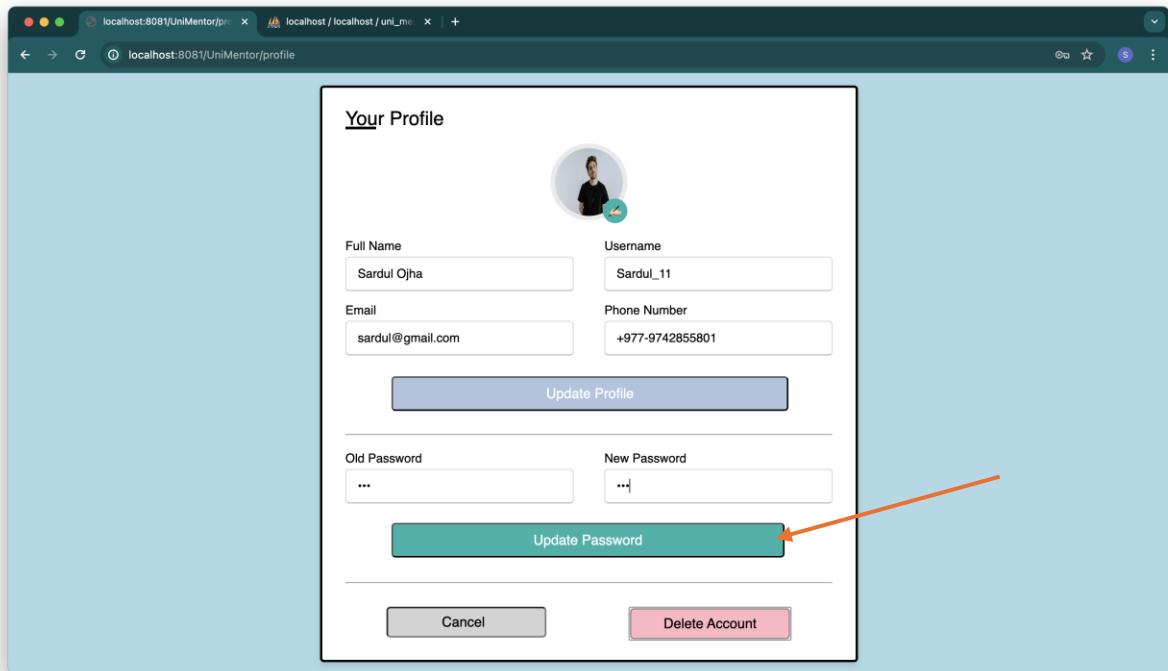
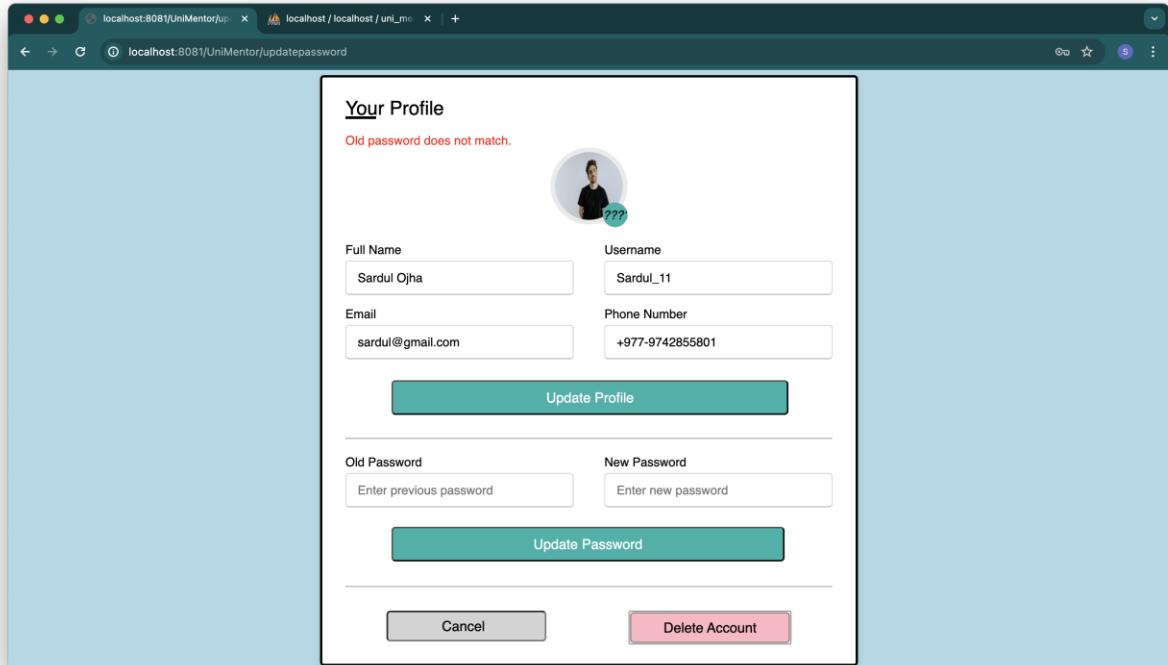


Figure 176: Screenshot of update password button clicked

5) Error message displayed:



Test 8.2. : Test for successful password reset

Objective	To update password after correct input.
Action	<ul style="list-style-type: none"> ➤ Profile page is clicked. ➤ Valid input is entered in password reset fields. ➤ Update password button is clicked. ➤ Logout button is clicked. ➤ Old password is entered to check. ➤ New password is again entered to check.
Expected Result	The password should be updated. Entering old password should display error message. Correct password should redirect to welcome page.
Actual Result	The password was updated successfully.
Conclusion	The test is successful.

Table 17: Test table for successful password reset

Proofs:

- 1) Clicking profile button on the navbar:

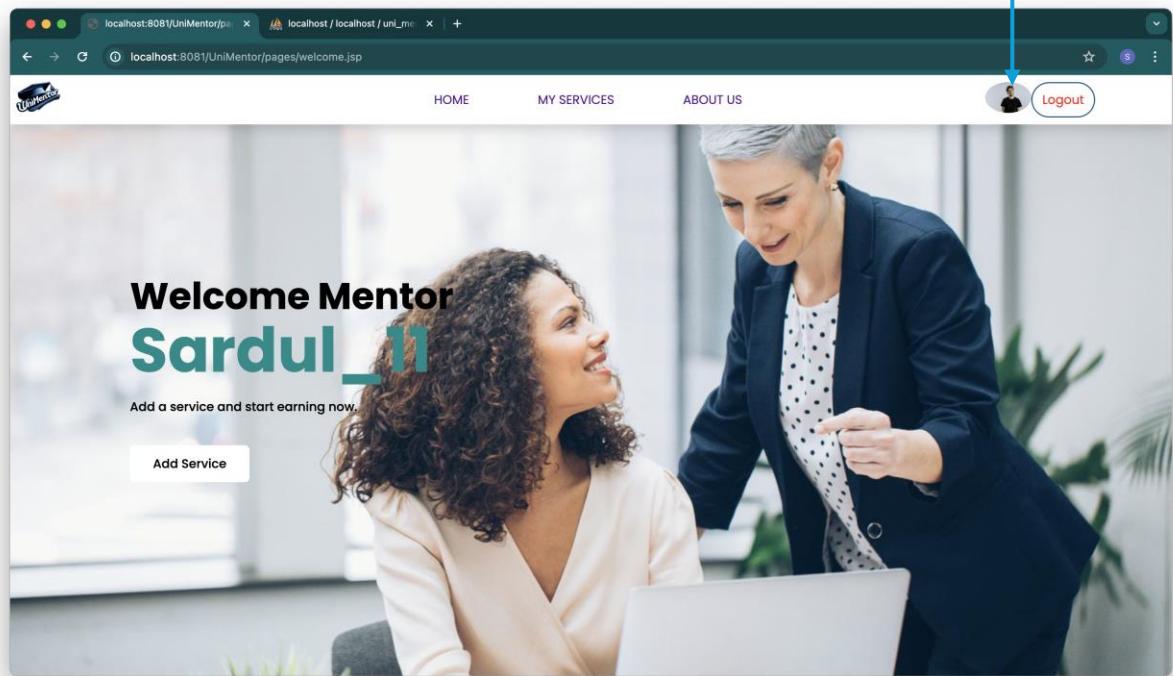


Figure 177: Screenshot of clicking profile button

2) Profile information displayed:

A screenshot of a web browser window displaying the 'profile' page of the UniMentor application. The page has a light blue background and features a central form titled 'Your Profile'. The form includes fields for 'Full Name' (Sardul Ojha), 'Username' (Sardul_11), 'Email' (sardul@gmail.com), and 'Phone Number' (+977-9742855801). There is also a circular profile picture placeholder. Below these fields is a teal 'Update Profile' button. Further down, there are fields for 'Old Password' (Enter previous password) and 'New Password' (Enter new password), separated by another teal 'Update Password' button. At the bottom of the form are two buttons: 'Cancel' (gray) and 'Delete Account' (pink).

Figure 178: Screenshot of profile information displayed

3) Entering correct input in update password fields:

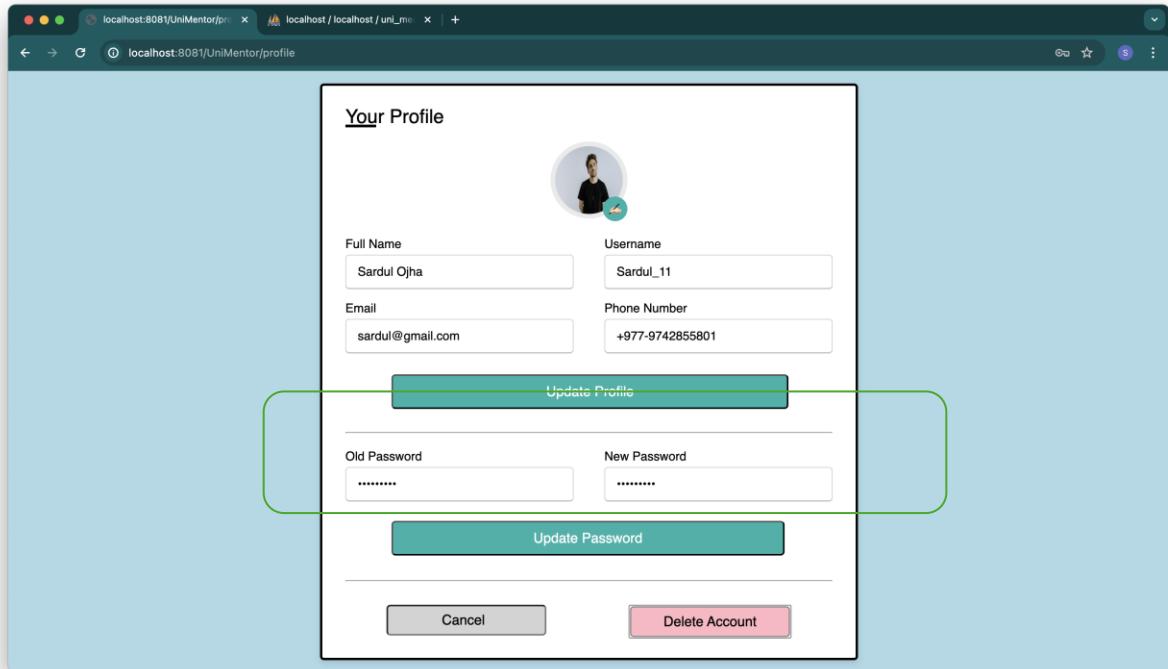


Figure 179: Screenshot of entering correct information in password fields

4) Clicking update password button:

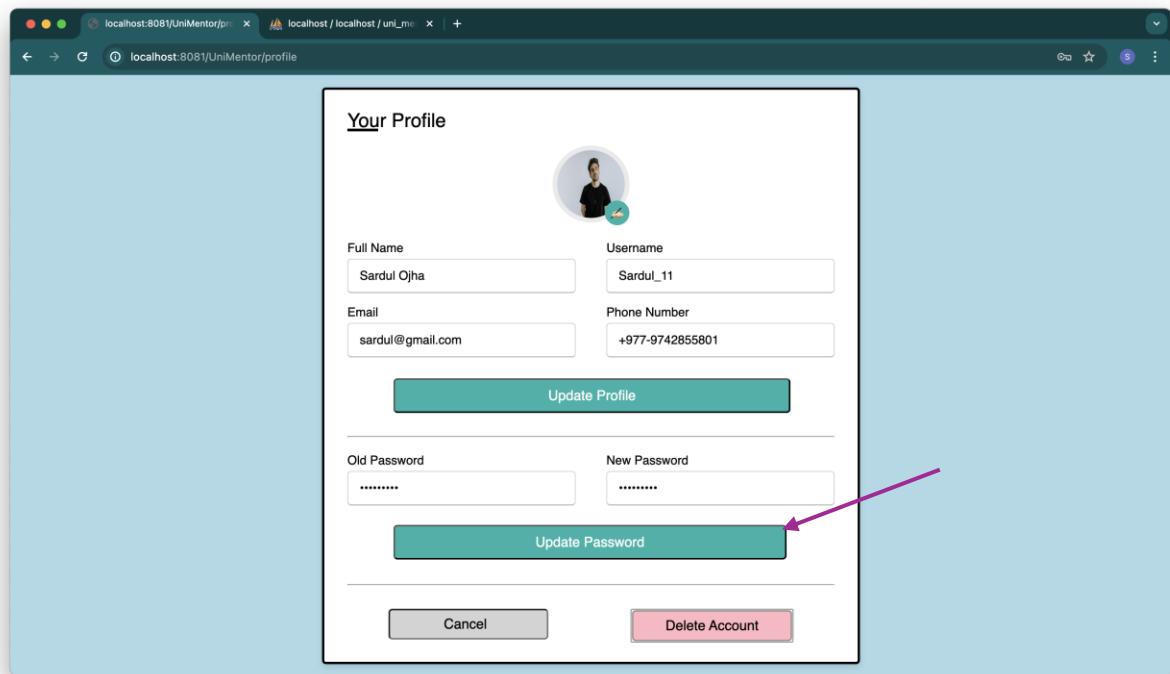


Figure 180: Screenshot of clicking update password button

5) Redirected to Welcome page:

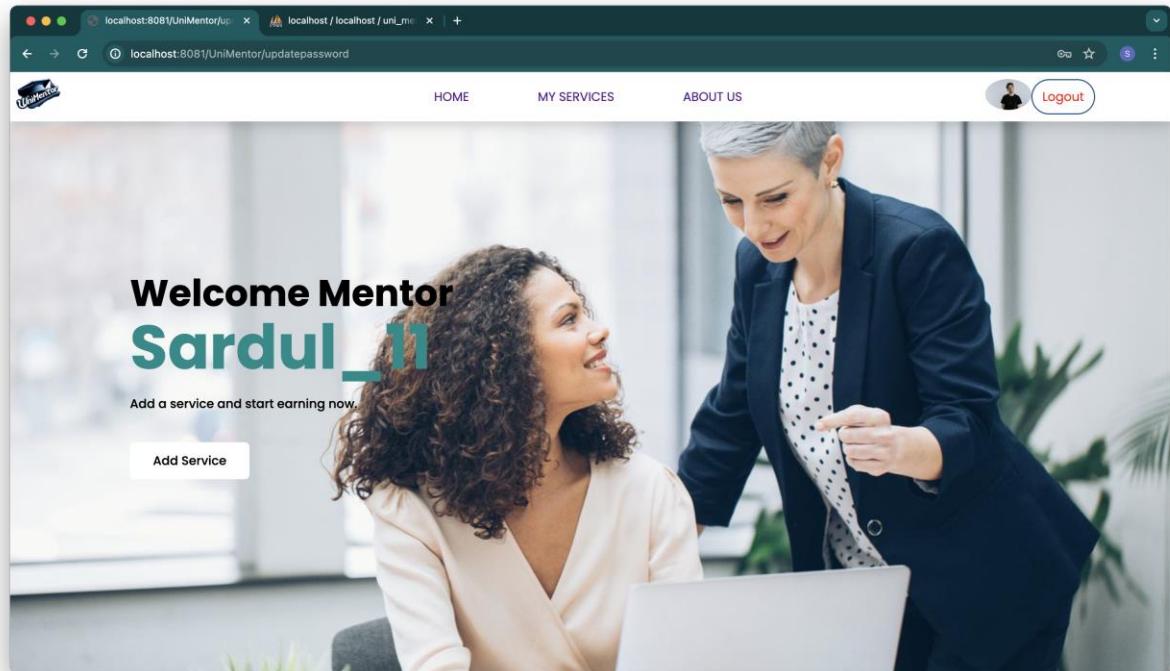


Figure 181: Screenshot of redirection to welcome page

6) Logging out for checking:

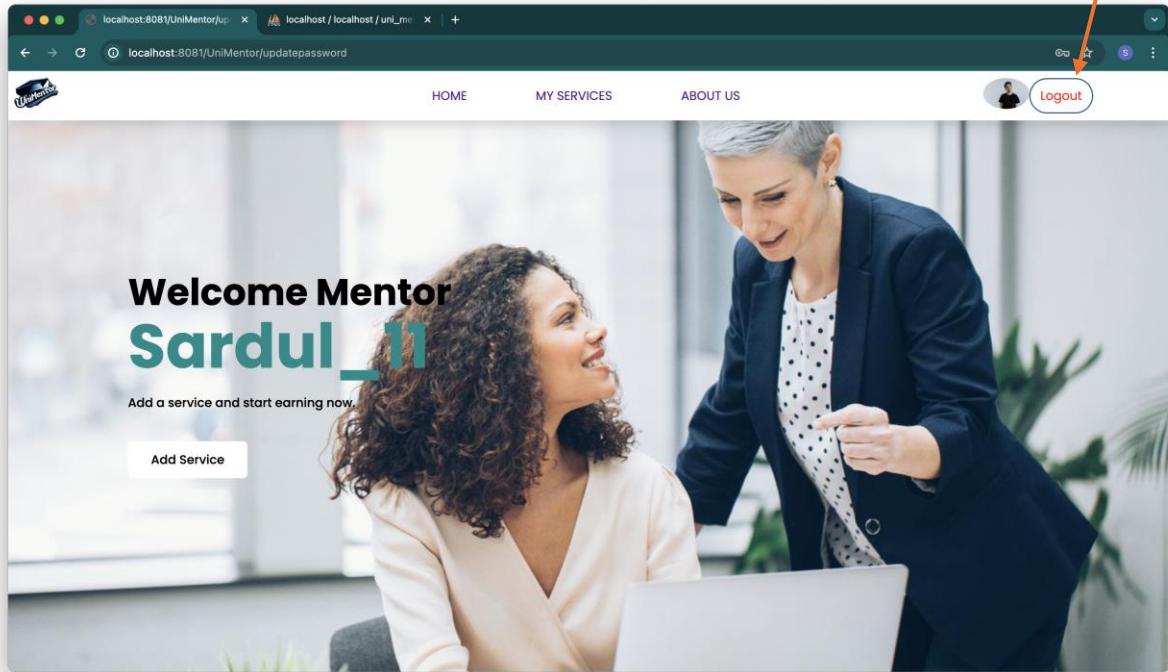


Figure 182: Screenshot of clicking logout button

7) Entering old password in login page:

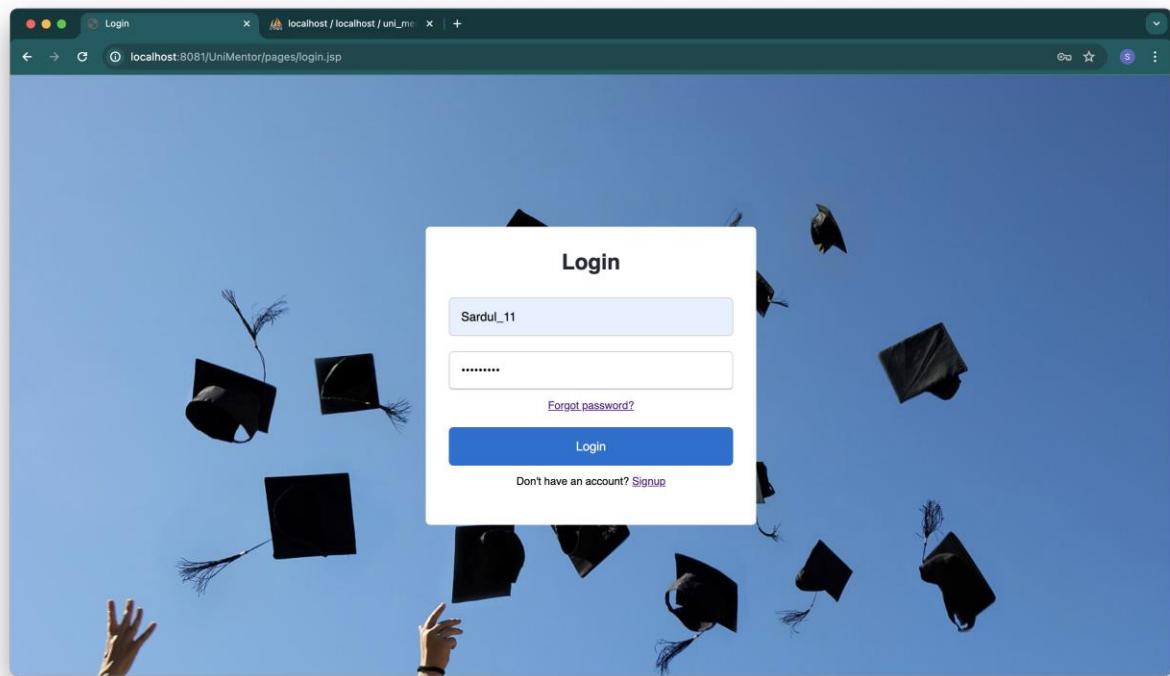


Figure 183: Screenshot of entering old password

8) Error message displayed:

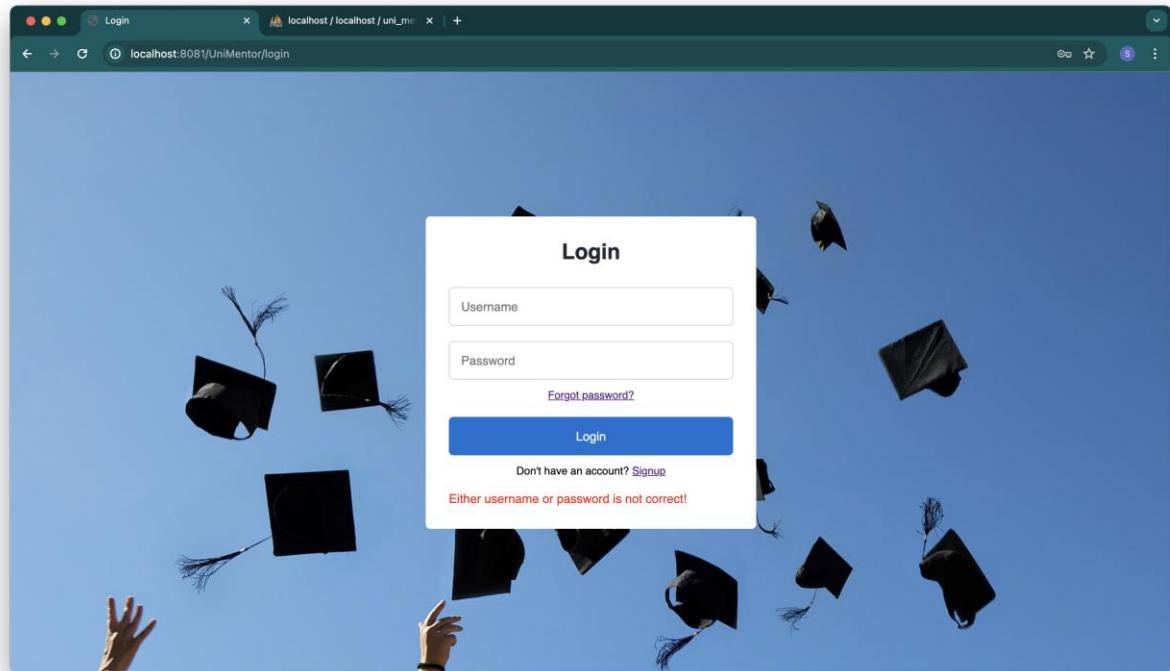


Figure 184: Screenshot of error message displayed

9) Entering correct password in login page:

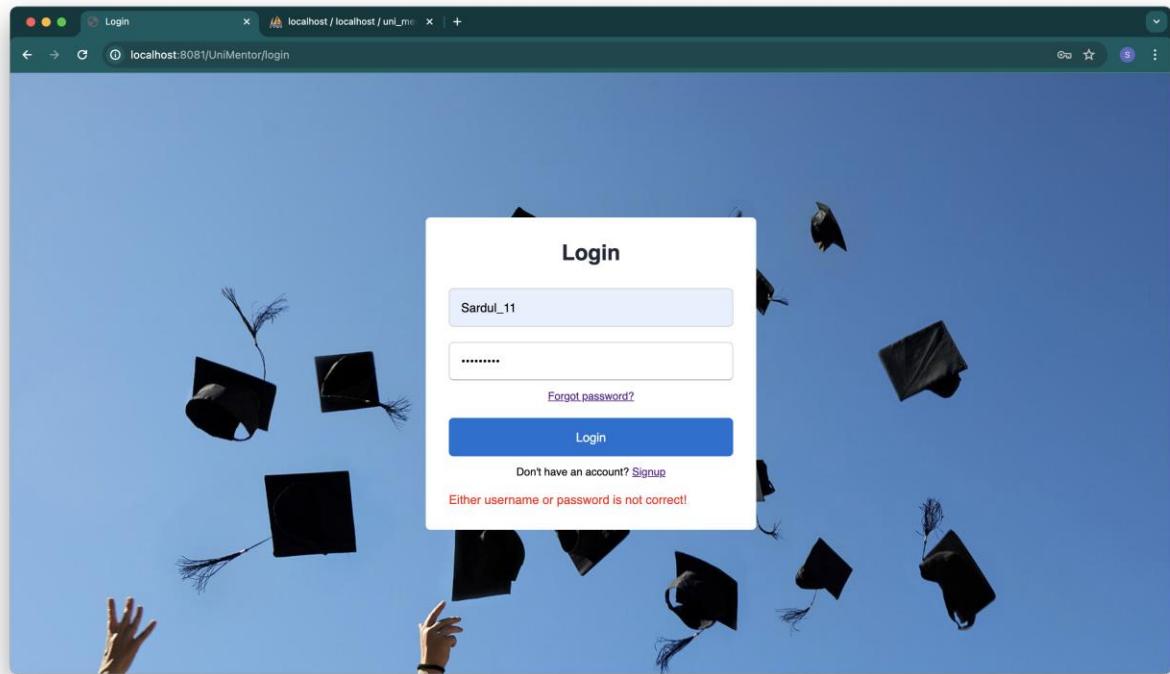


Figure 185: Screenshot of entering correct password

10) Welcome page displayed (Login success):

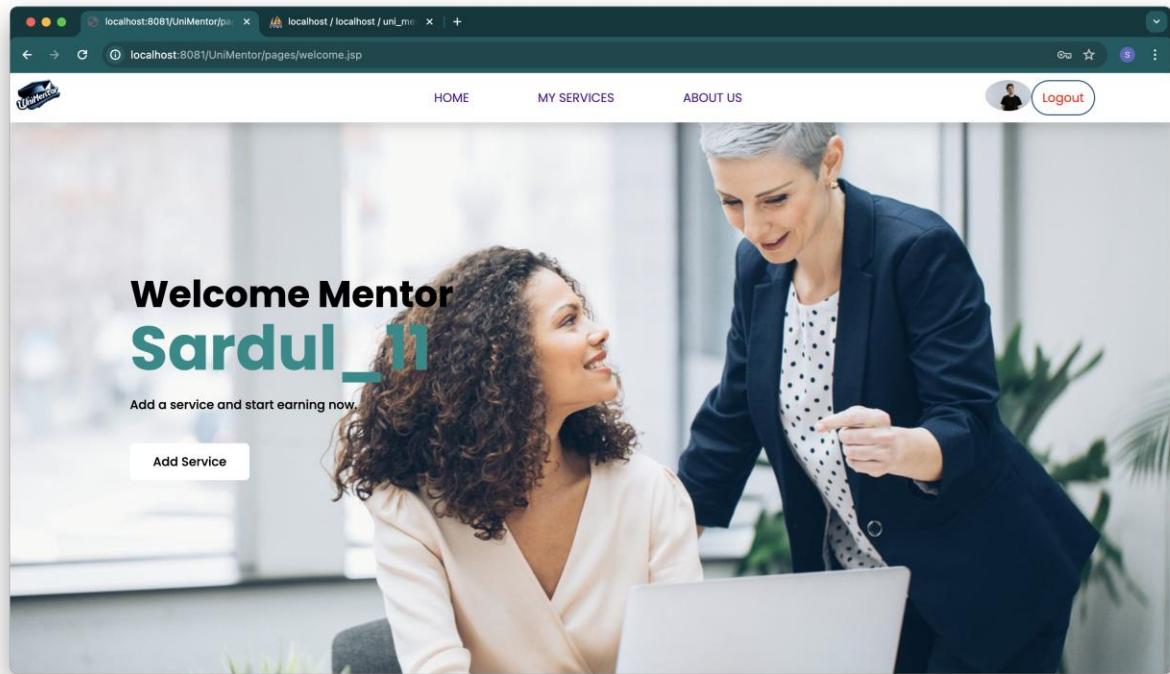


Figure 186: Screenshot of login success

5.9. Test 9: Test for Search

Test 9.1. : Test for search

Objective	To display searched services.
Action	<ul style="list-style-type: none"> ➤ Search value is entered in textbox in service page. ➤ Search button is clicked.
Expected Result	All the services with search value should be displayed.
Actual Result	The searched services were displayed.
Conclusion	The test is successful.

Table 18: Test table for search

Proofs:

- 1) Entering search value in searchbox:

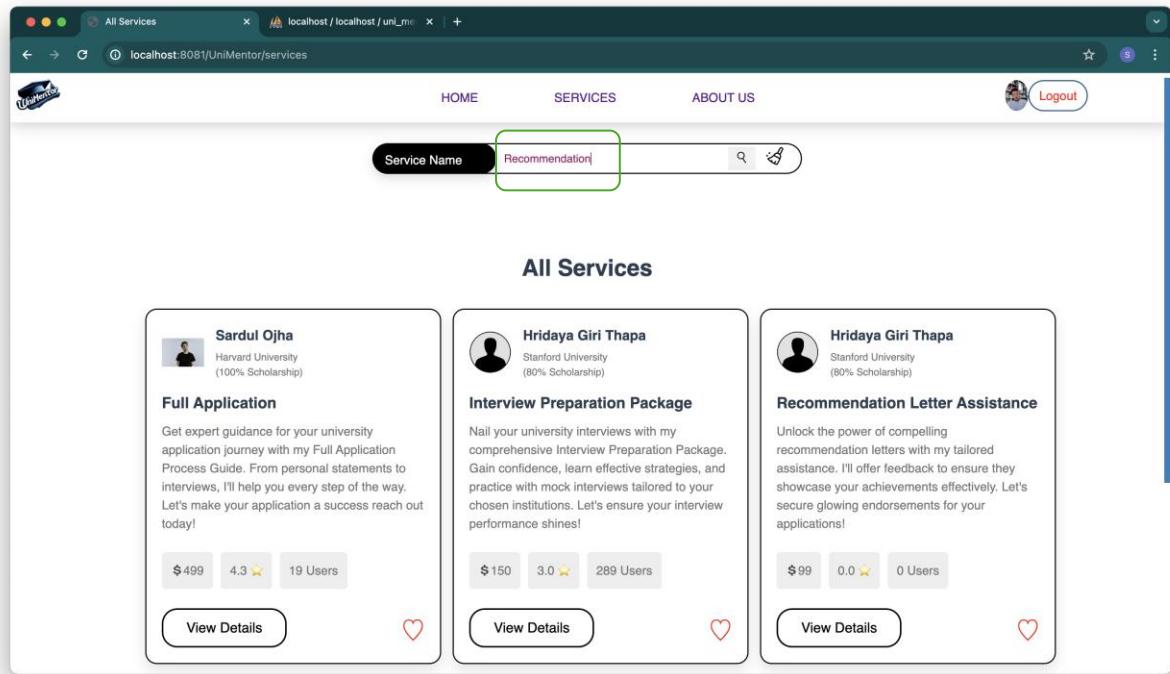


Figure 187: Screenshot of entering search value

2) Search button clicked:

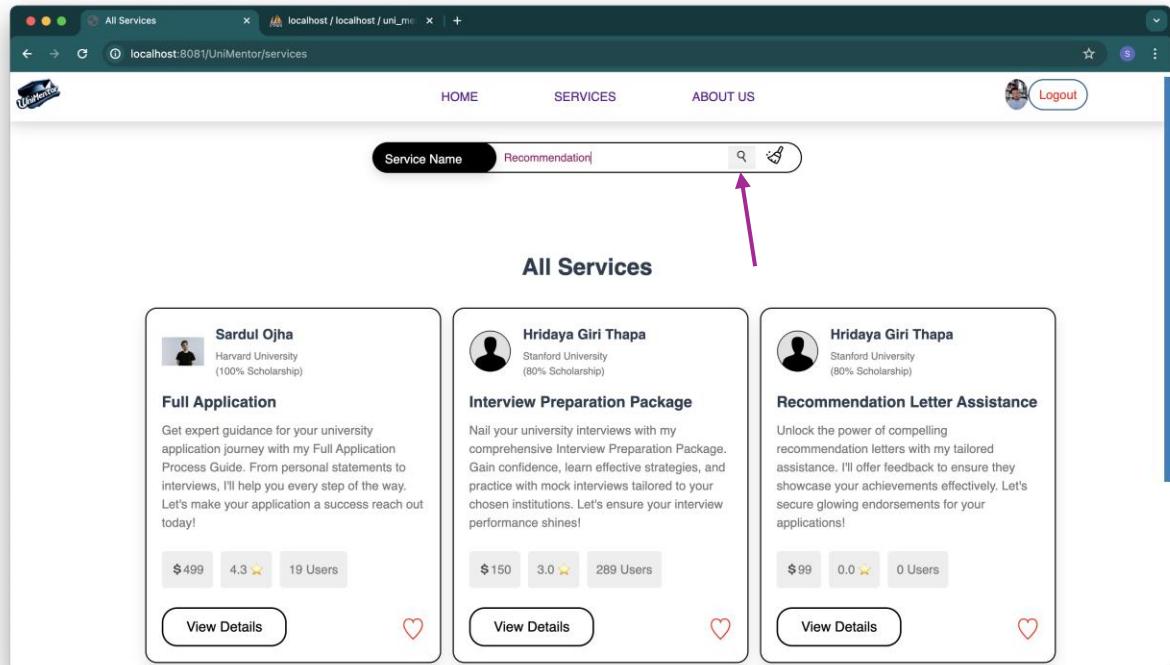


Figure 188: Screenshot of search button clicked

3) Search results displayed:

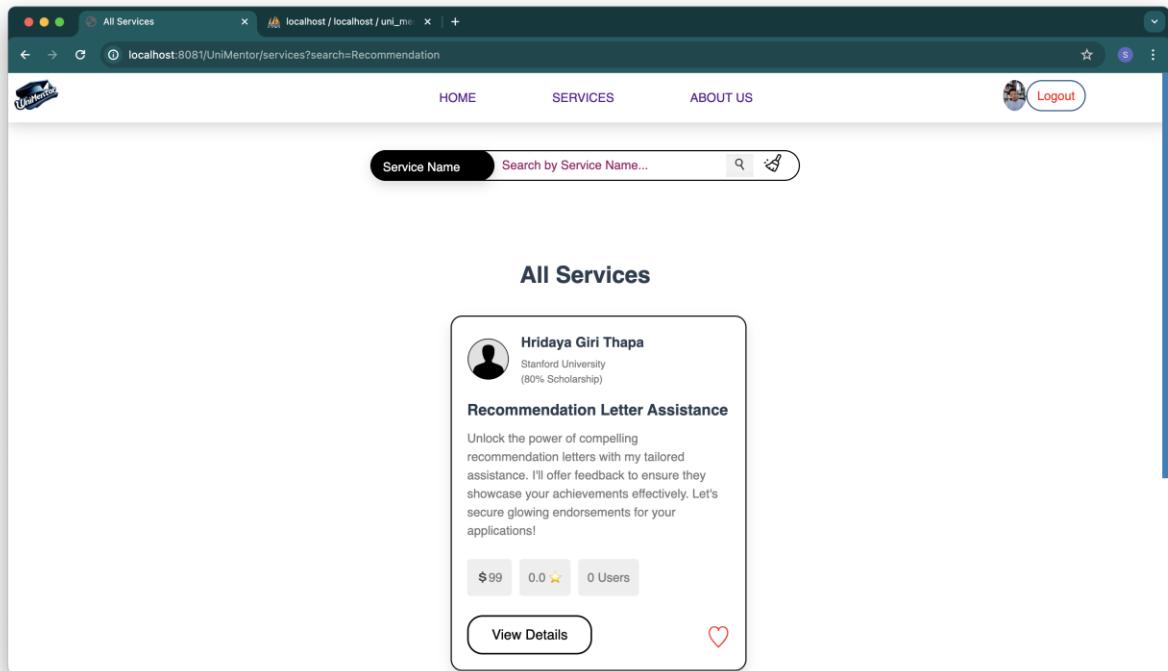


Figure 189: Screenshot of search results displayed

Test 9.2. : Test for added service being searched

Objective	To display new added service when searched.
Action	<ul style="list-style-type: none"> ➤ Add button is clicked to add a service. ➤ A new service is added. ➤ Added service name is entered in textbox in service page. ➤ Search button is clicked.
Expected Result	The new added service should be displayed.
Actual Result	The new service was displayed.
Conclusion	The test is successful.

Table 19: : Test table for added service search

Proofs:

1) Clicking add button in my service page:

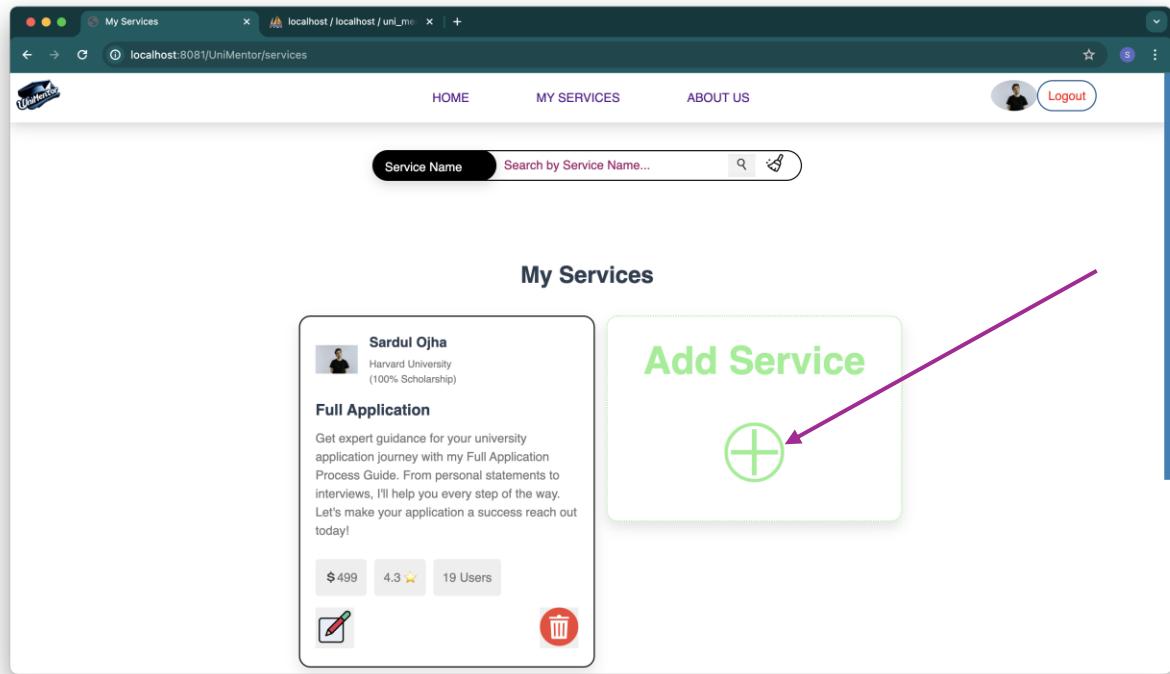


Figure 190: Screenshot of clicking add button

2) Adding new service:

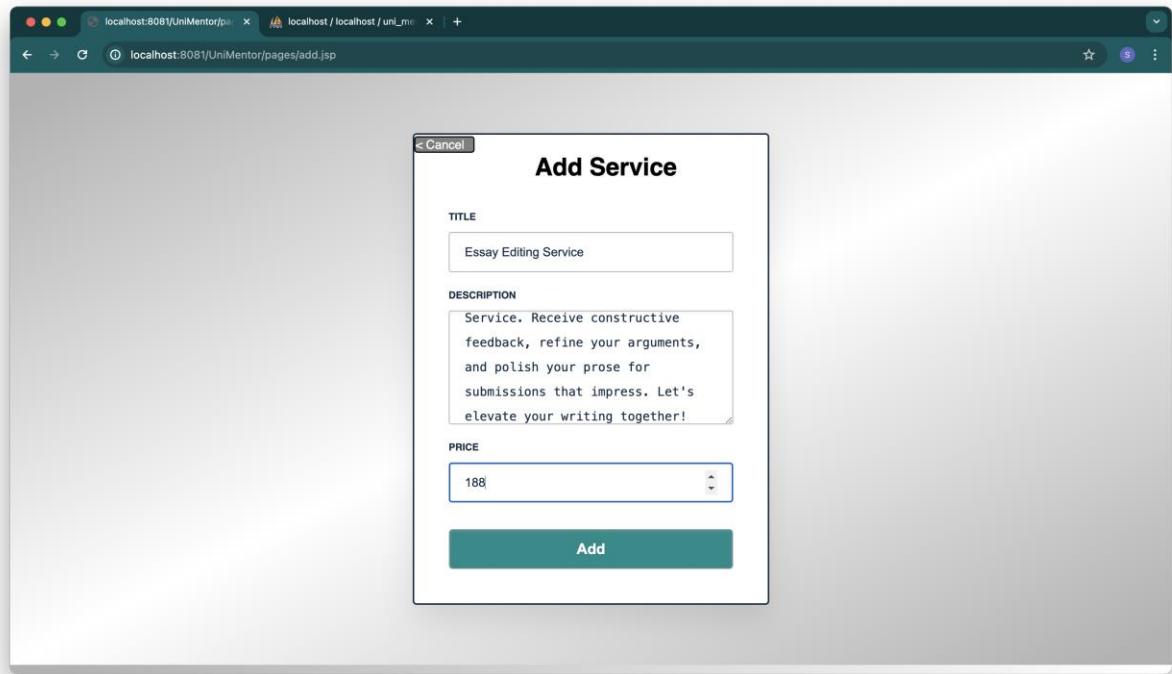


Figure 191: Screenshot of adding new service

3) Entering added service name in searchbox:

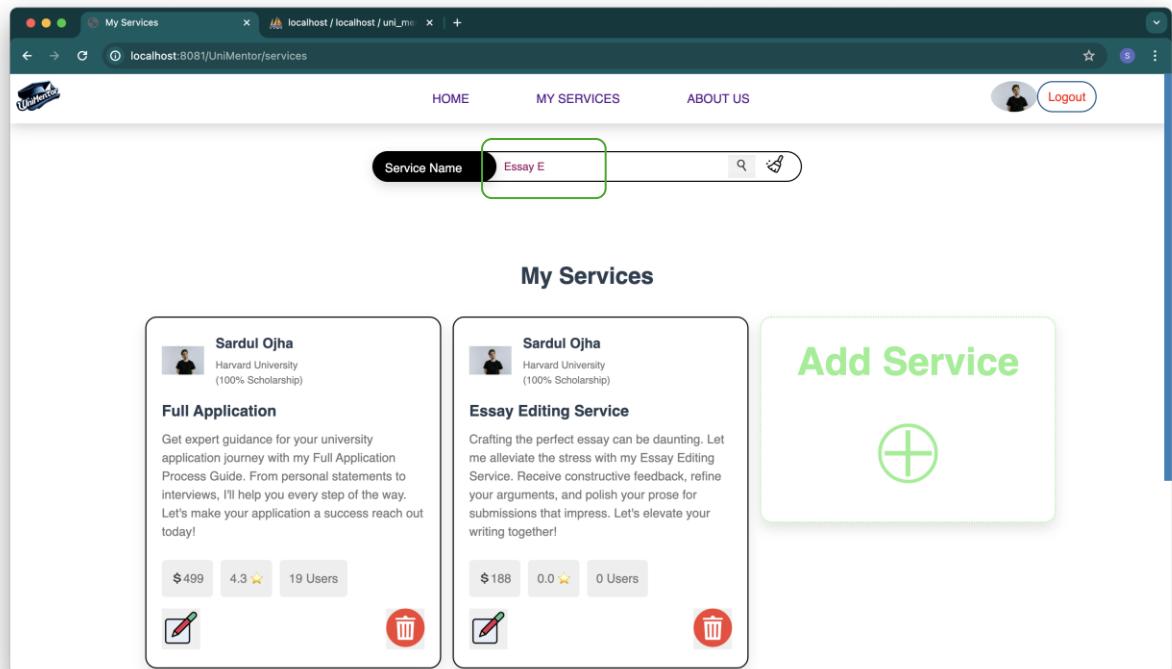


Figure 192: Screenshot of entering added service name

4) Search button clicked:

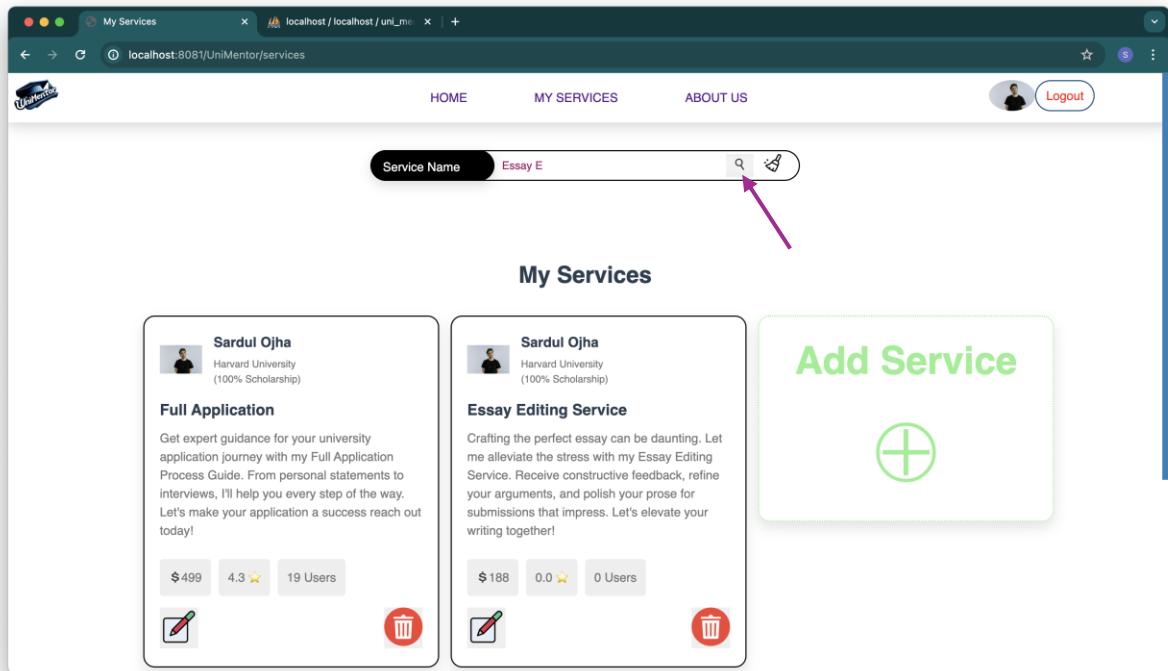


Figure 193: Screenshot of search button clicked

5) Added service displayed:

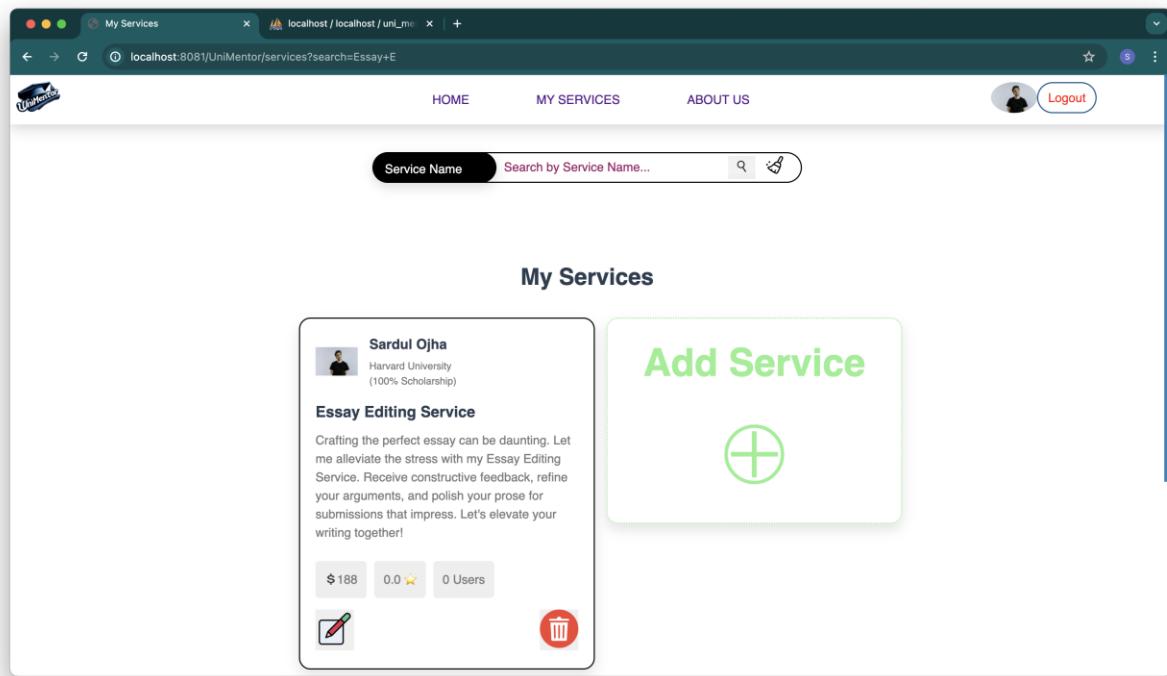


Figure 194: Screenshot of added service displayed

Test 9.3. : Test for deleted items not being searched

Objective	To not display the deleted service.
Action	<ul style="list-style-type: none"> ➤ Delete button is clicked to delete the selected service. ➤ Yes button is clicked to confirm delete. ➤ The deleted service name is entered in textbox in service page. ➤ Search button is clicked.
Expected Result	The deleted service should not be displayed.
Actual Result	The deleted service was not displayed.
Conclusion	The test is successful.

Table 20: Test table for deleted service search

Proofs:

- 1) Clicking delete button in for the selected service:

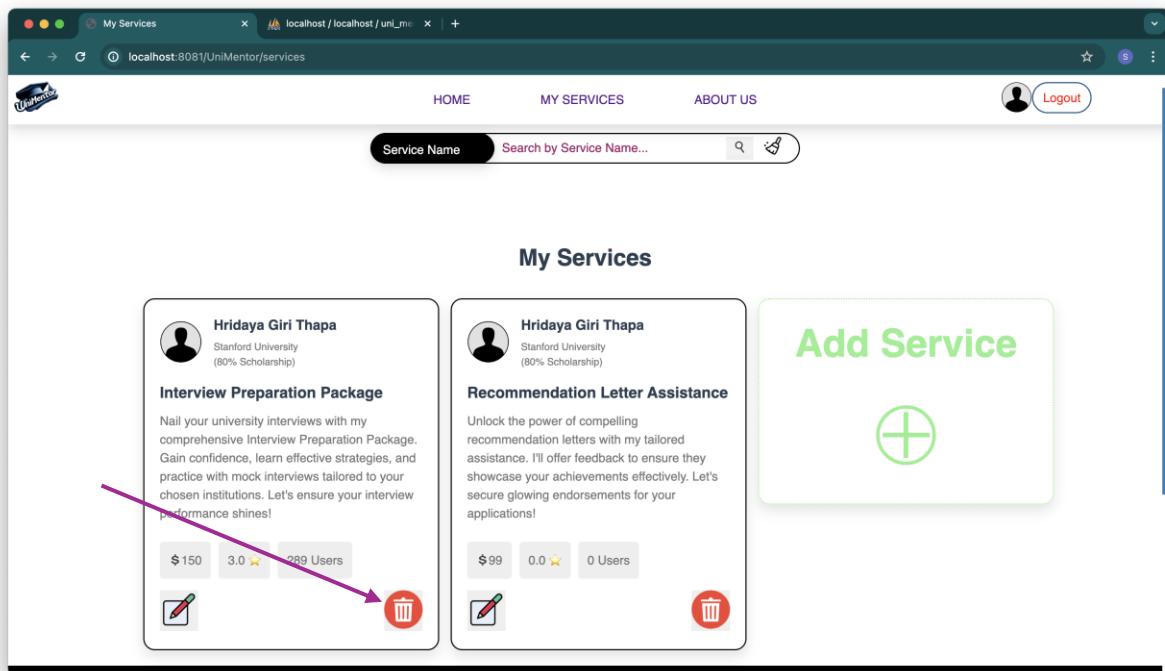


Figure 195: Screenshot of clicking delete button

- 2) Clicking yes to confirm delete:

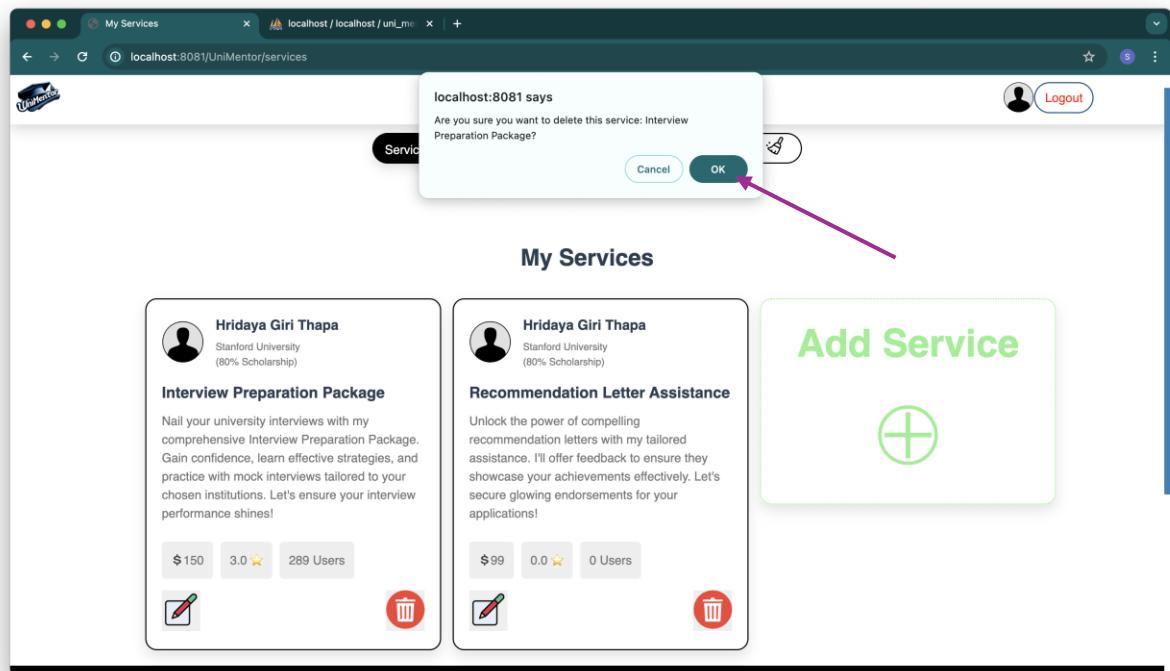


Figure 196: Screenshot of clicking ok button

3) Entering deleted service name in searchbox:

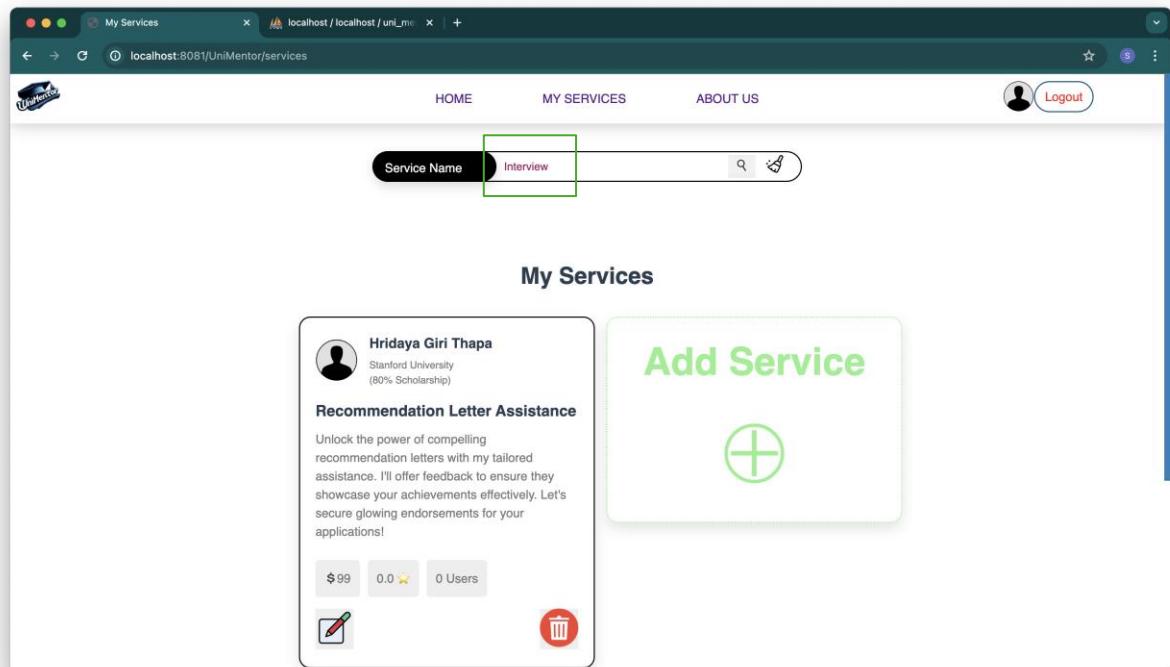


Figure 197: Screenshot of entering deleted service title

4) Search button clicked:

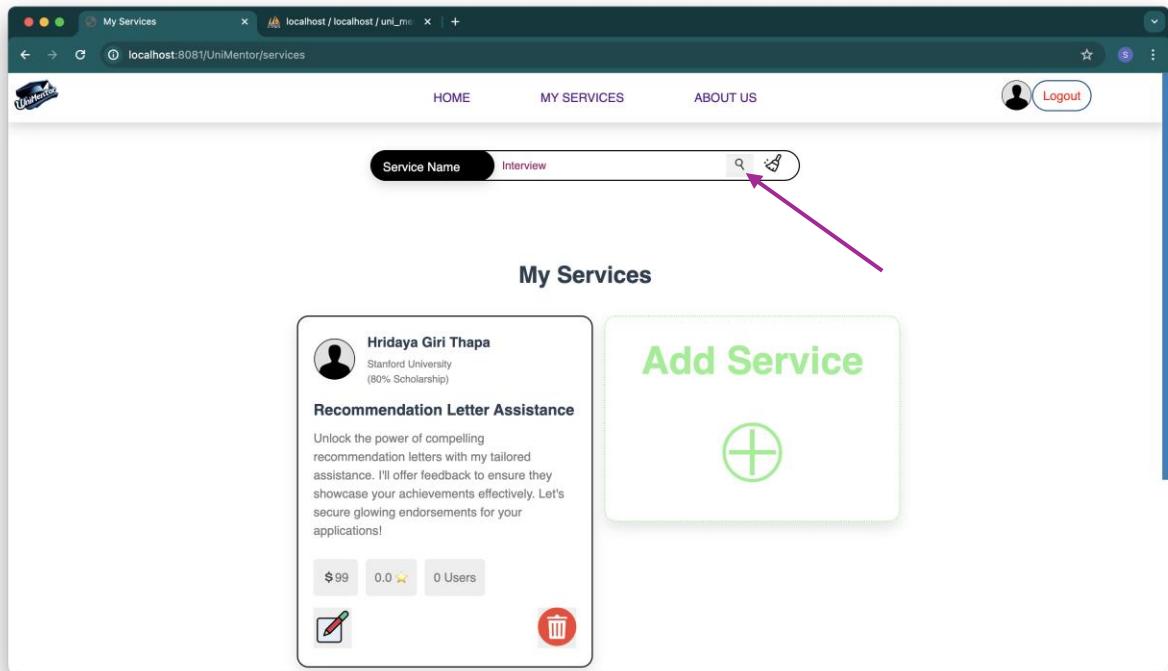


Figure 198: Screenshot of search button clicked

5) Deleted service not displayed:

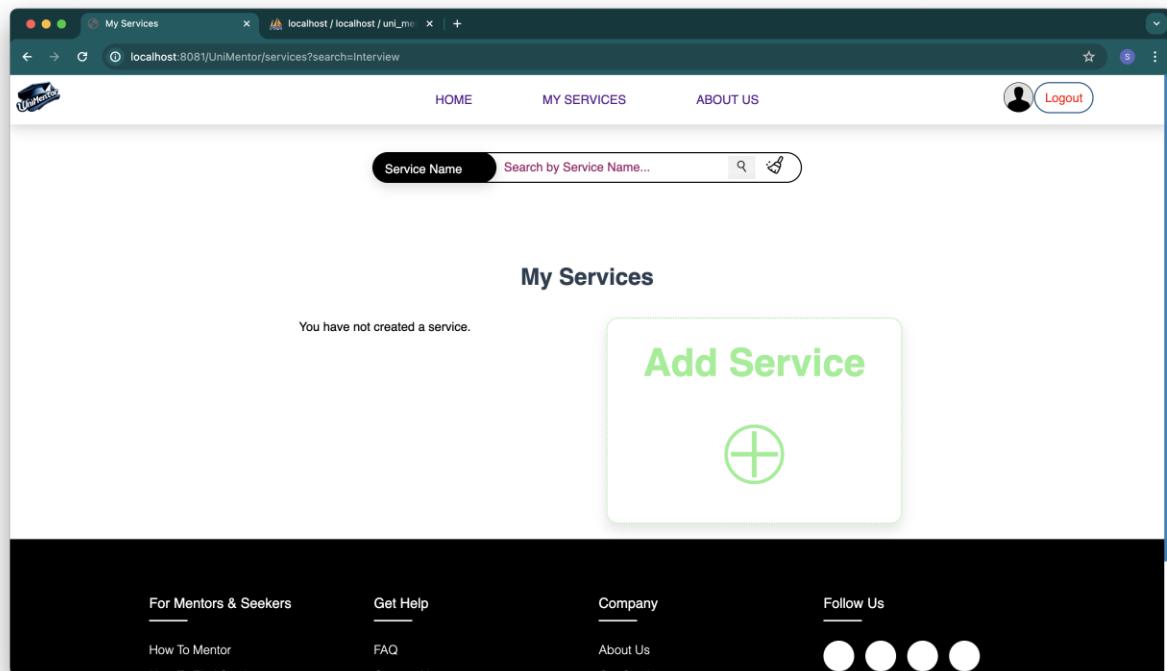


Figure 199: Screenshot of no service found

Test 9.4. : Test for updated item being searched

Objective	To display the updated service when searched.
Action	<ul style="list-style-type: none"> ➤ Update button is clicked to update the selected service. ➤ The service title is updated. ➤ Old title is entered in the textbox in service page. ➤ Updated service title is entered in textbox in service page. ➤ Search button is clicked.
Expected Result	The updated service should be displayed whereas the old service title should display no services.

Actual Result	The old service was not displayed, and the new title was displayed.
Conclusion	The test is successful.

Table 21: Test table for updated service search

Proofs:

- 1) Clicking update button for the selected service:

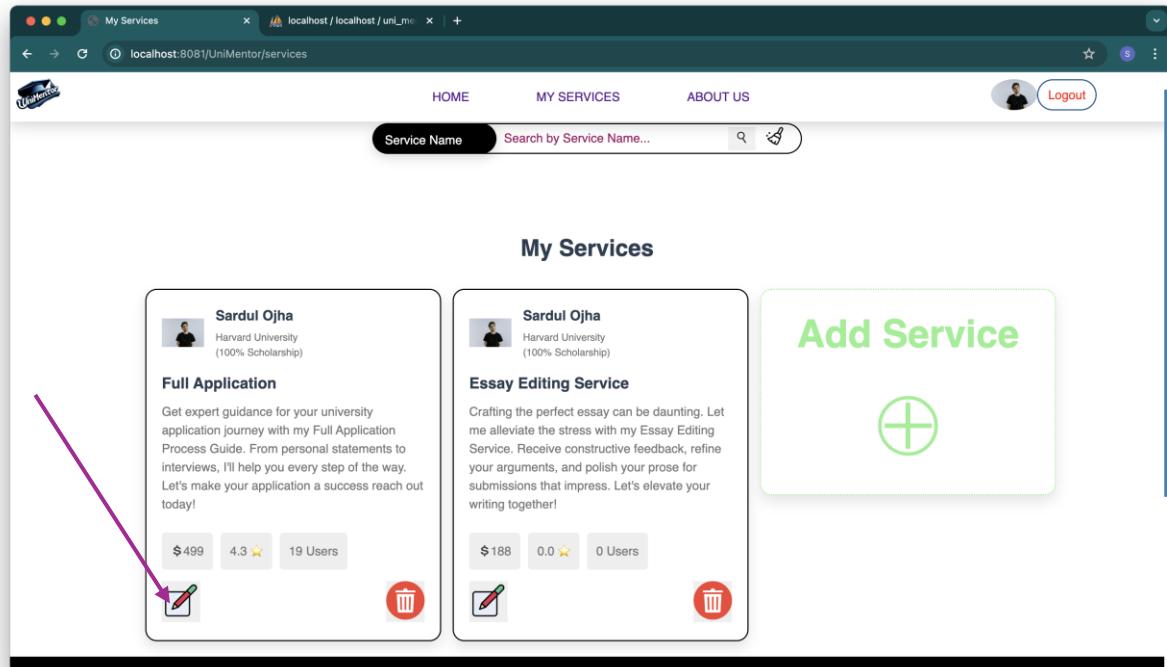


Figure 200: Screenshot of clicking update button

- 2) Updating title of the service:

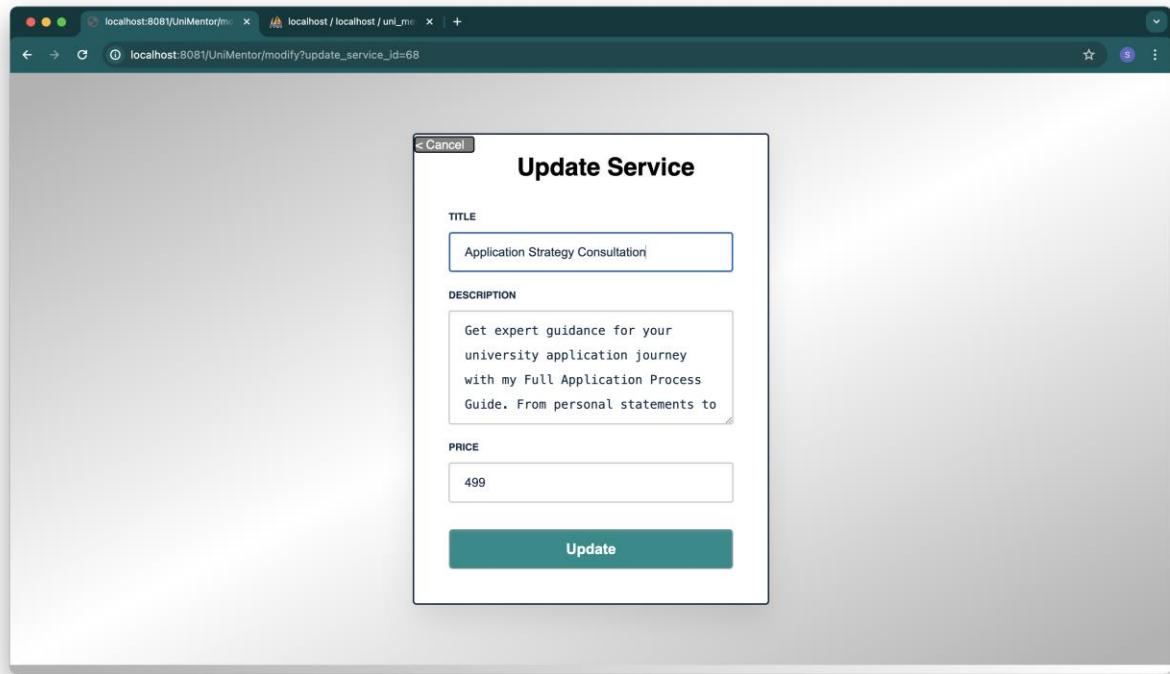


Figure 201: Screenshot of updating title

3) Entering old service title in searchbox and search button clicked:

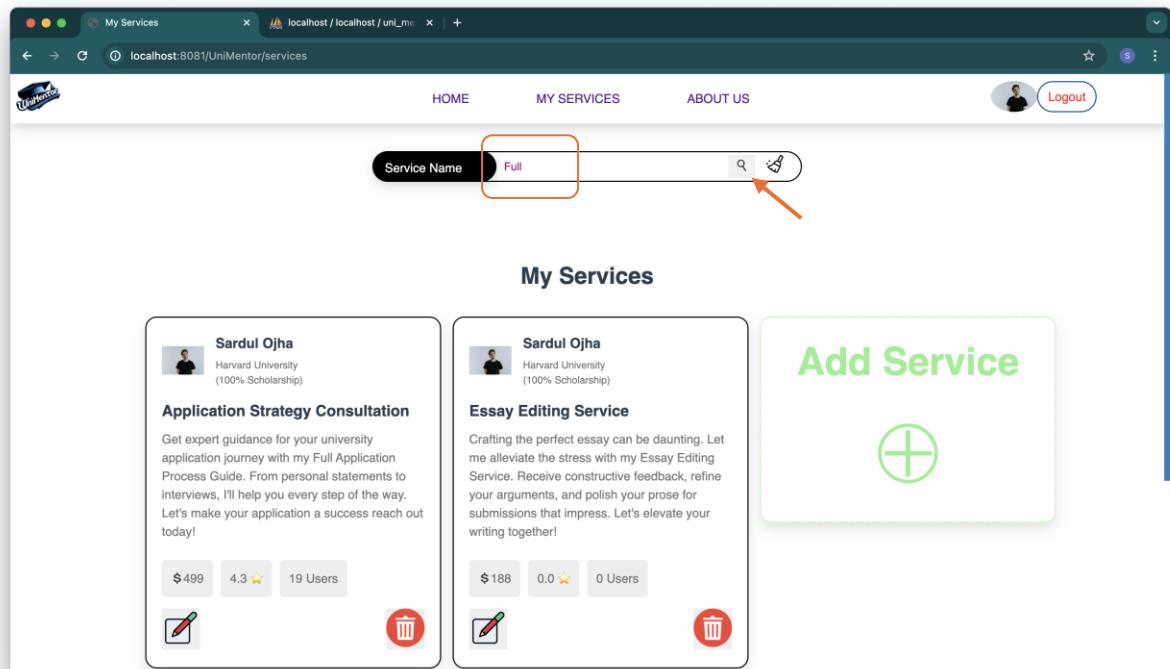


Figure 202: Screenshot of entering old title in searchbox

4) Service not displayed:

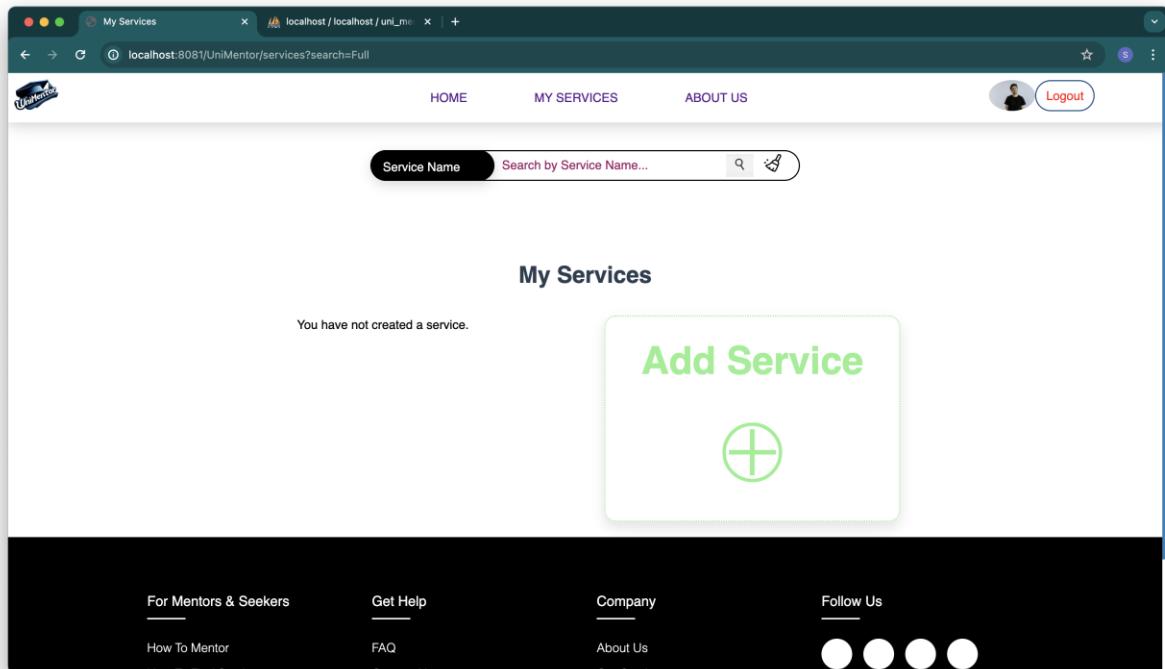


Figure 203: Screenshot of old service title not displayed

5) Entering updated service name in searchbox and search button clicked:

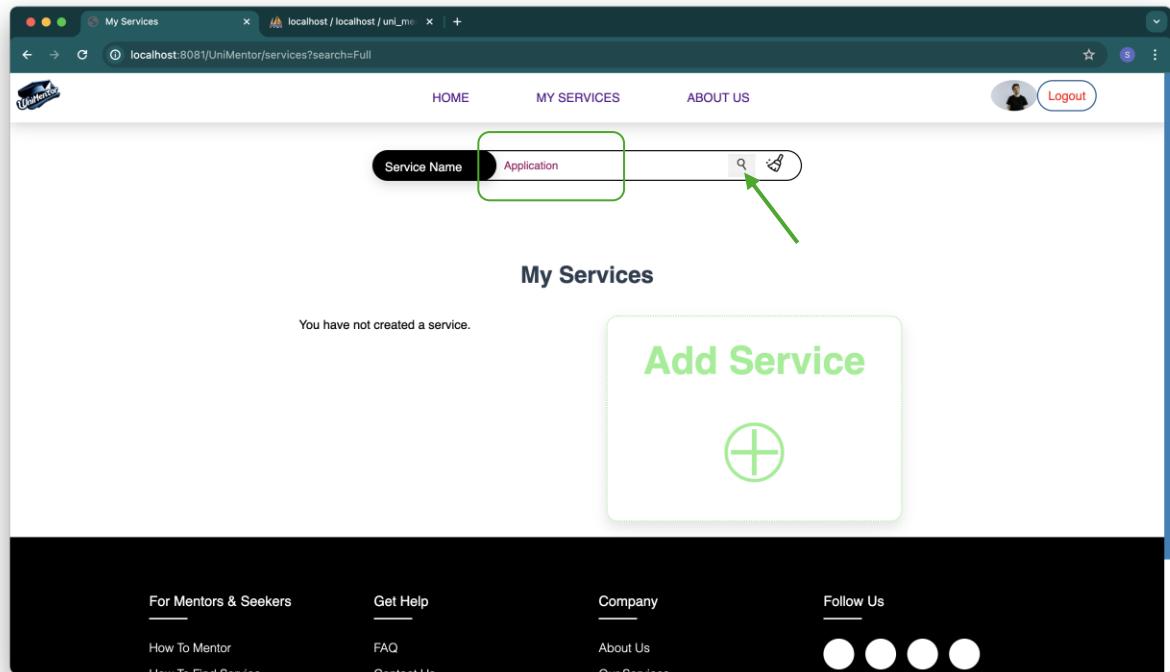


Figure 204: Screenshot of entering updated title

6) Service displayed:

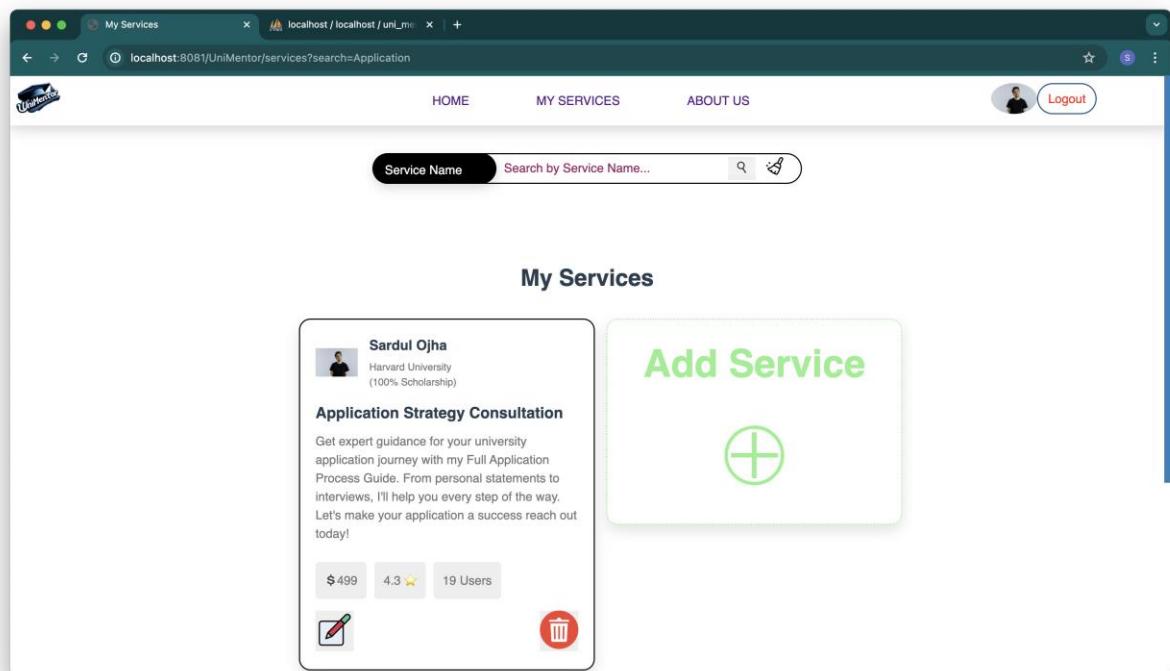


Figure 205: Screenshot of updated service displayed

5.10. Test 10: Test for navigation (Authentication Filter)

Test 10.1. : Test for access of logged out users

Objective	To navigate logged out users to login page when trying to access pages like services, profile, welcome.
Action	When logged out, <ul style="list-style-type: none"> ➤ Service is clicked in nav bar to access service page. ➤ Profile page URL is given in address bar. ➤ Welcome page URL is given in address bar.
Expected Result	Each action should navigate user to login page.
Actual Result	The protected pages were not accessed, and user was navigated to login page.
Conclusion	The test is successful.

Table 22: Test table for page access to logged out users

Proofs:

- 1) Clicking service in nav bar to access service page:

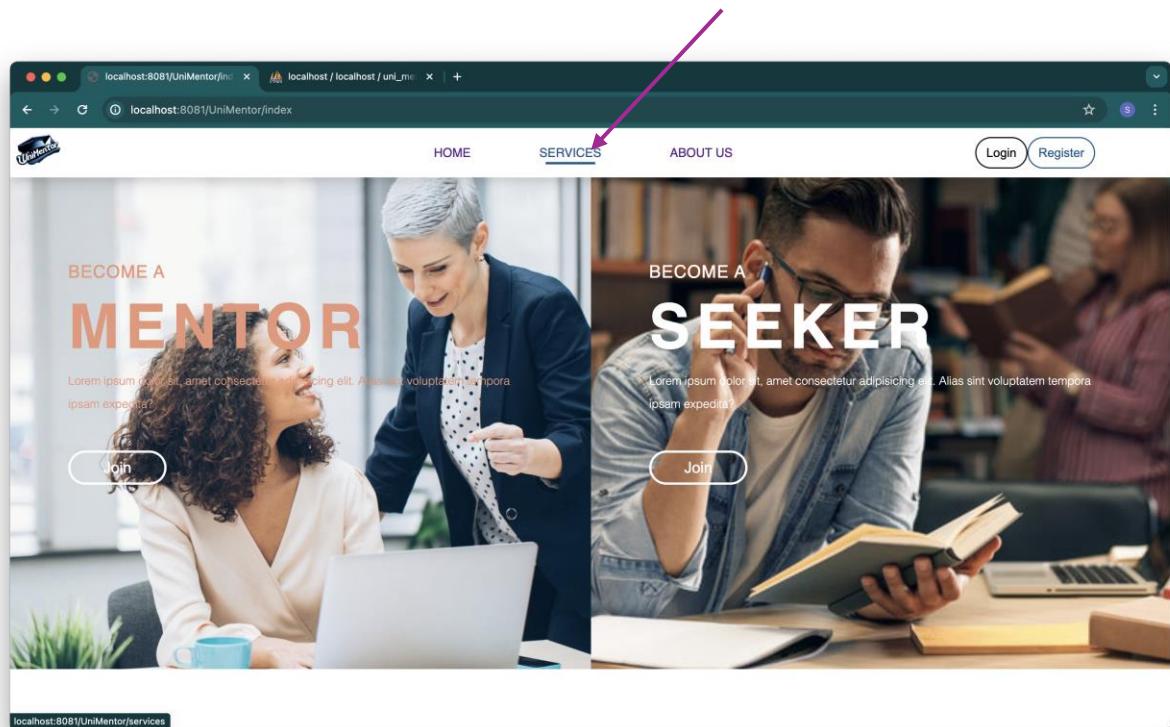


Figure 206: Screenshot of clicking services in nav bar

2) Login page displayed:

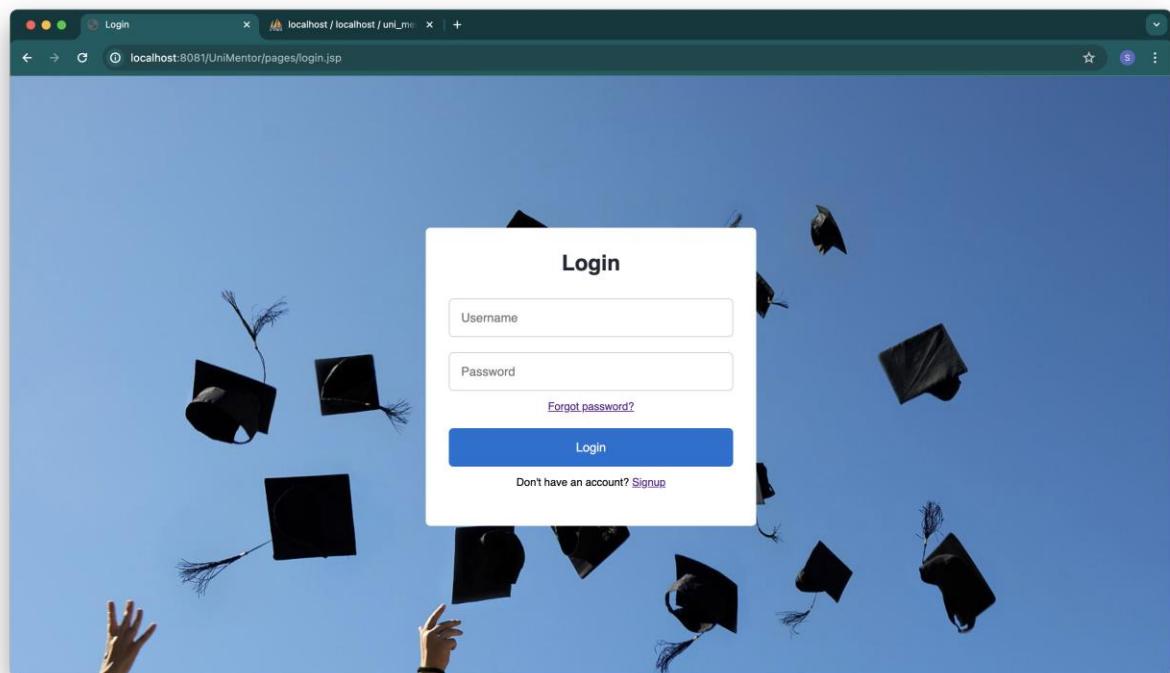


Figure 207: Screenshot of login page displayed

3) Giving profile page URL in address bar:

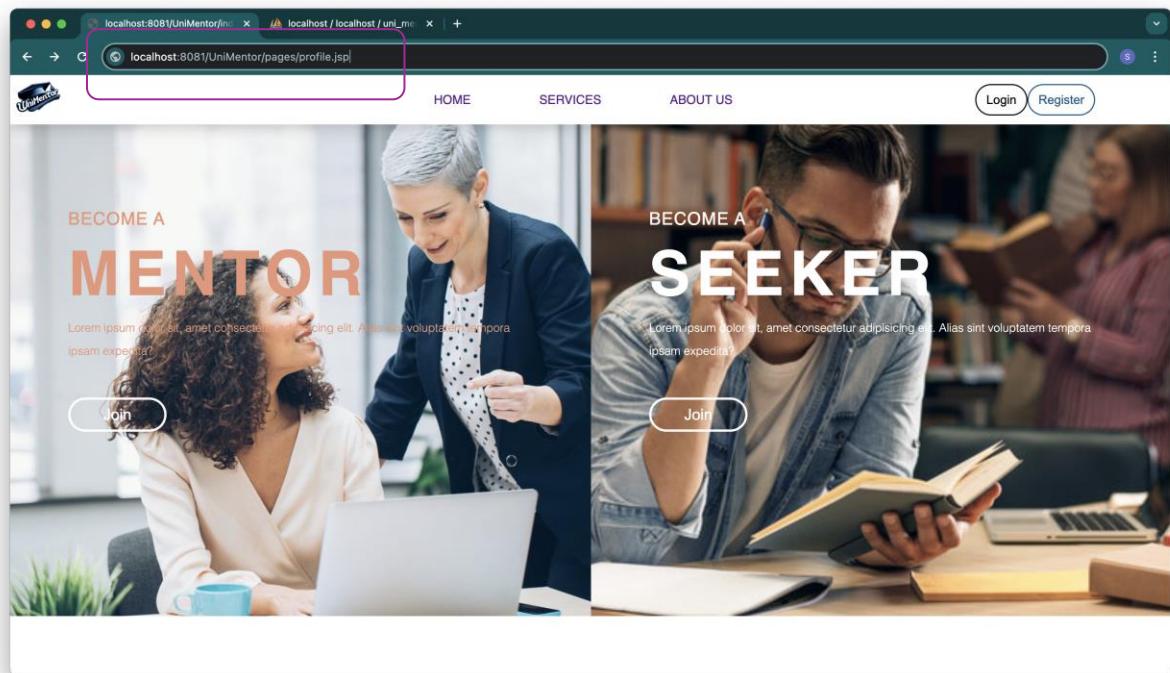


Figure 208: Screenshot of giving profile page url

4) Login page displayed:

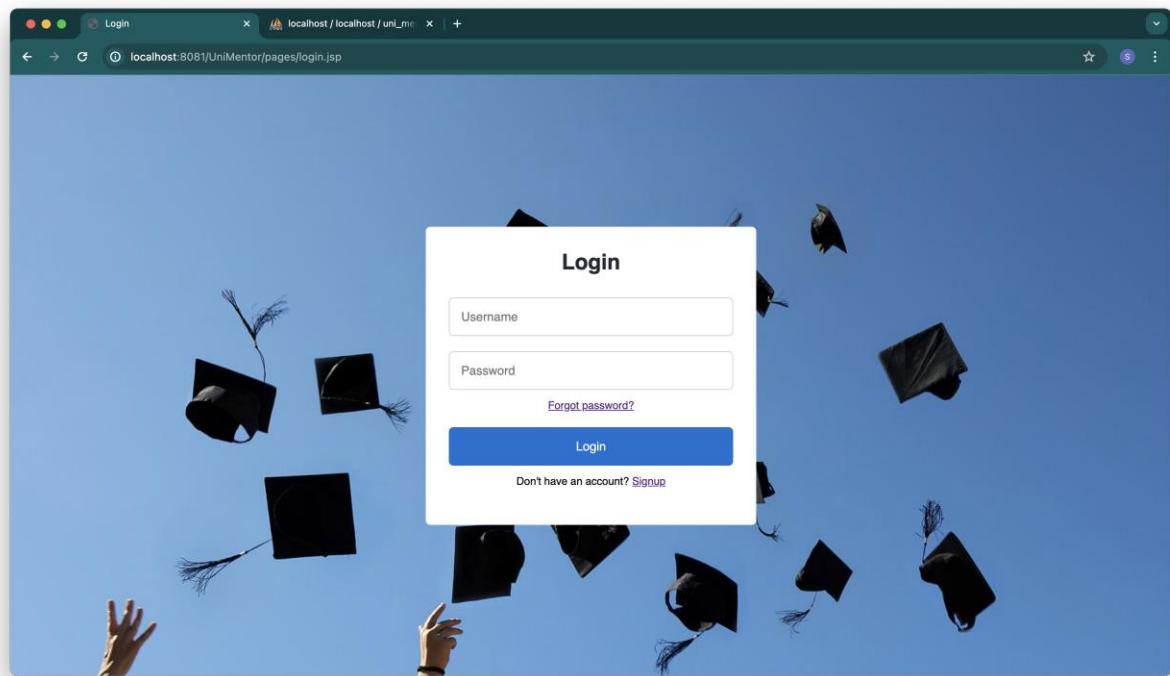


Figure 209: Screenshot of login page displayed

5) Giving welcome page URL in address bar:

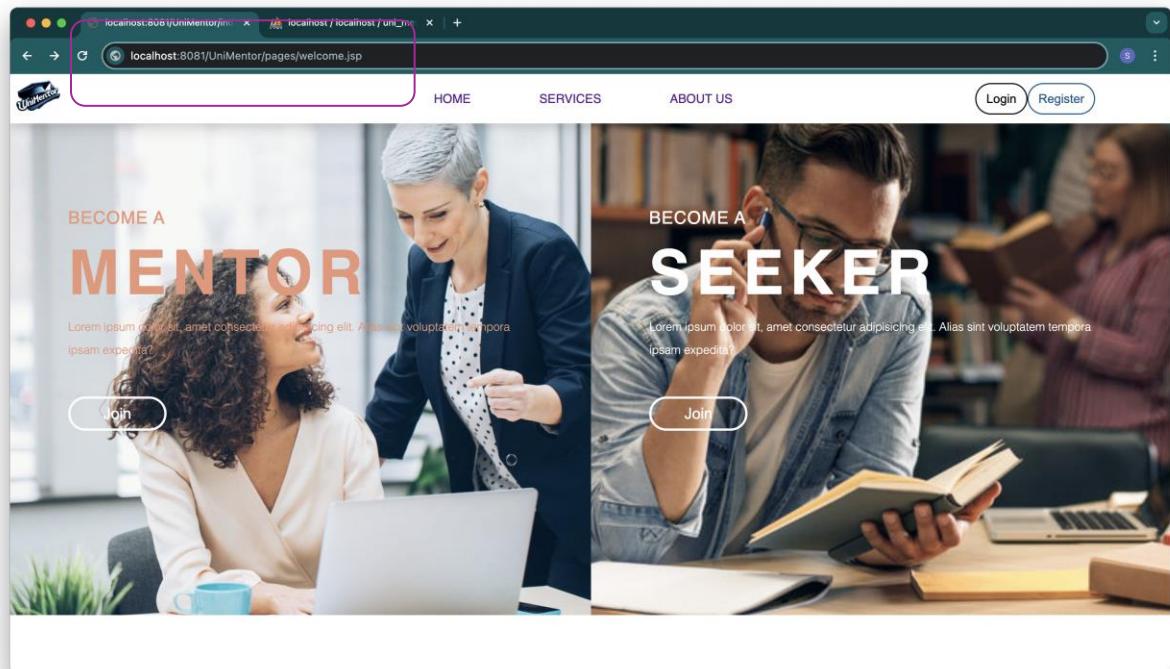


Figure 210: Screenshot of giving welcome page url

6) Login page displayed:

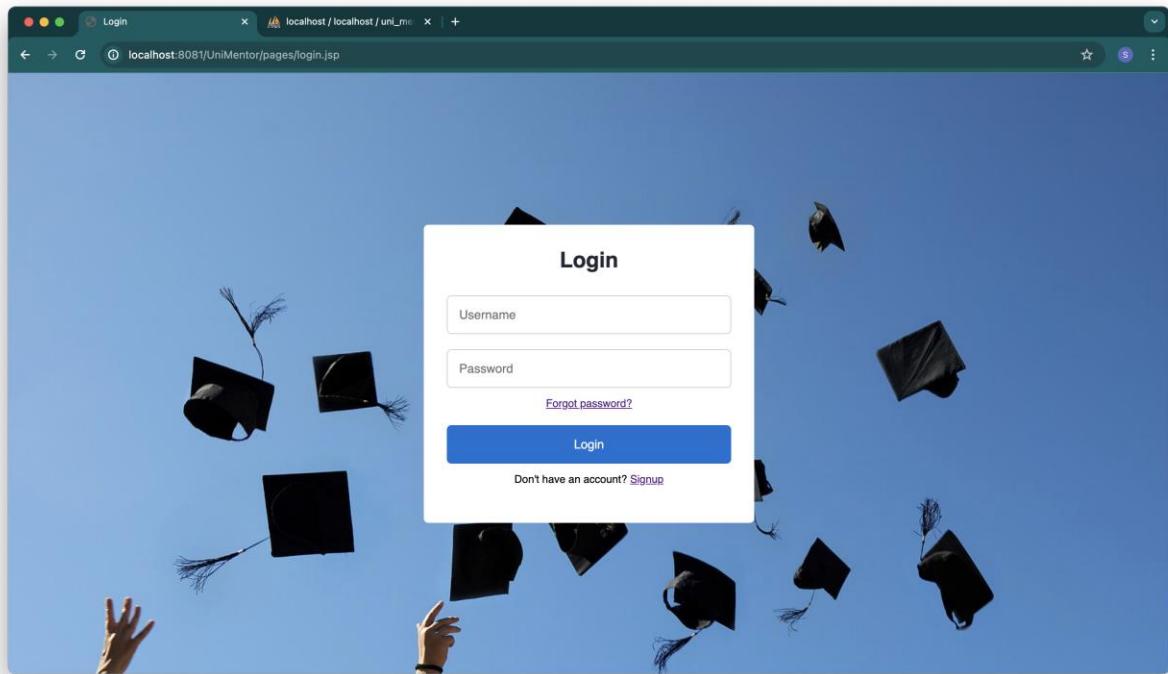


Figure 211: Screenshot of login page displayed

Test 10.2. : Test for access for logged in users

Objective	To navigate logged in users to welcome page when trying to access pages like login and register.
Action	When logged in, <ul style="list-style-type: none"> ➤ Login page URL is given in address bar. ➤ Register page URL is given in address bar.
Expected Result	Each action should navigate user to welcome page.
Actual Result	The protected pages were not accessed, and user was navigated to welcome page.
Conclusion	The test is successful.

Table 23: Test table for page access to logged in users

Proofs:

- 1) Logging in as user (mentor or seeker):

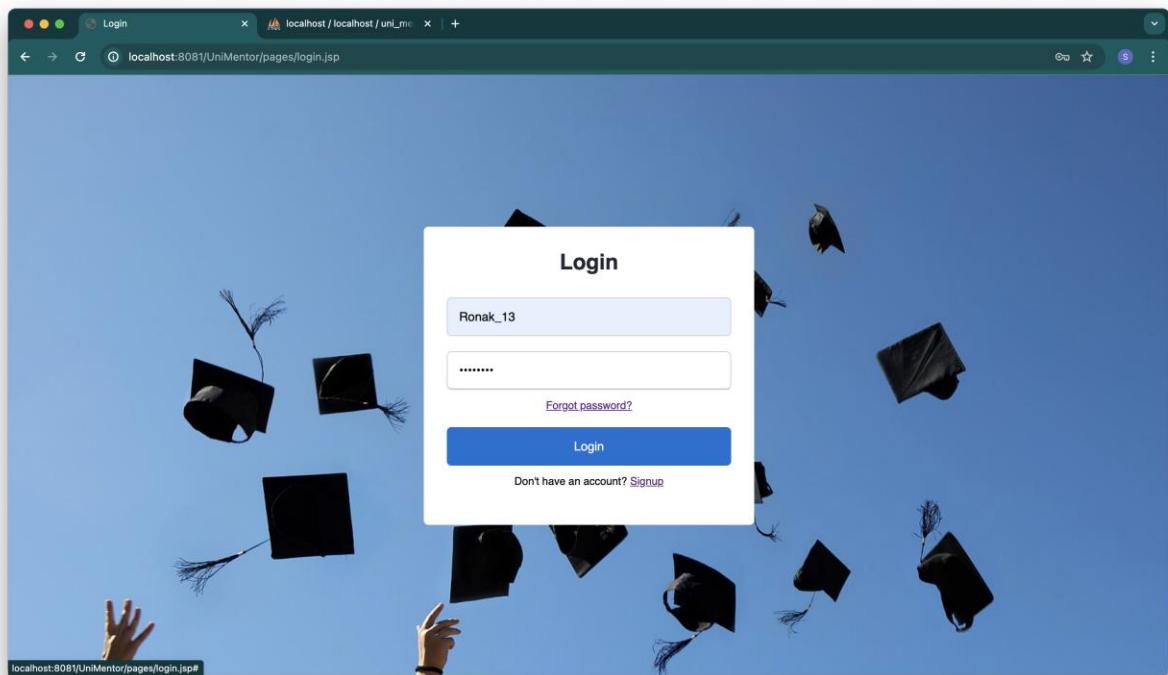


Figure 212: Screenshot of logging in as user

- 2) Welcome page displayed:

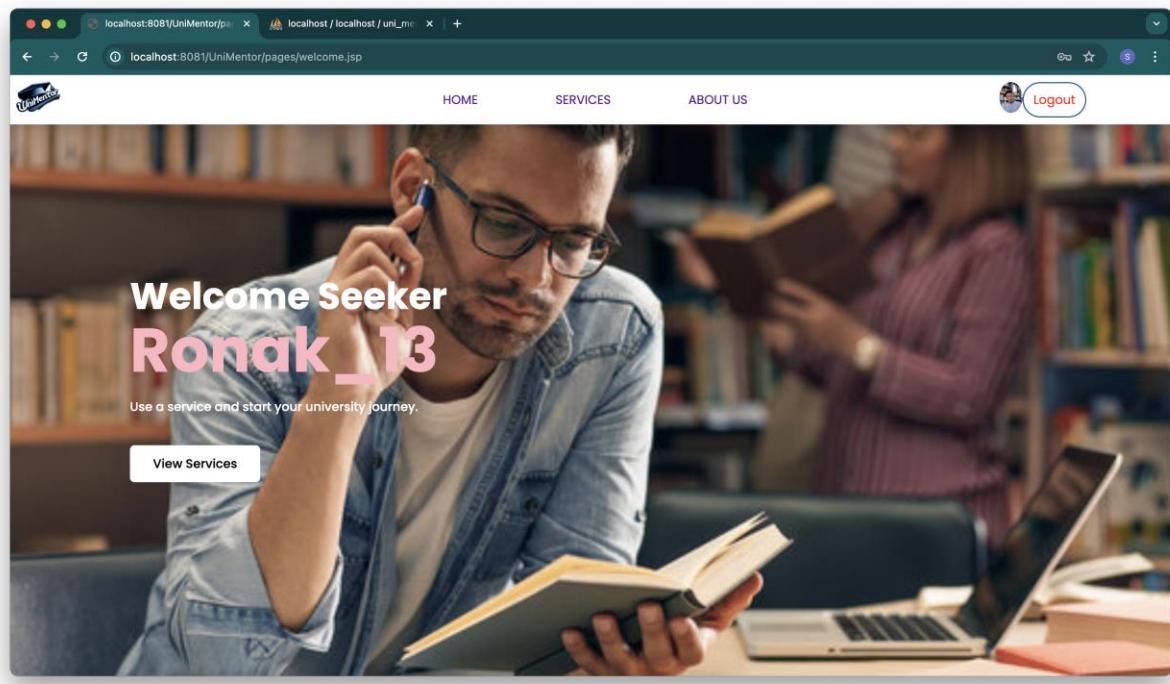


Figure 213: Screenshot of welcome page displayed

3) Giving login page URL in address bar:

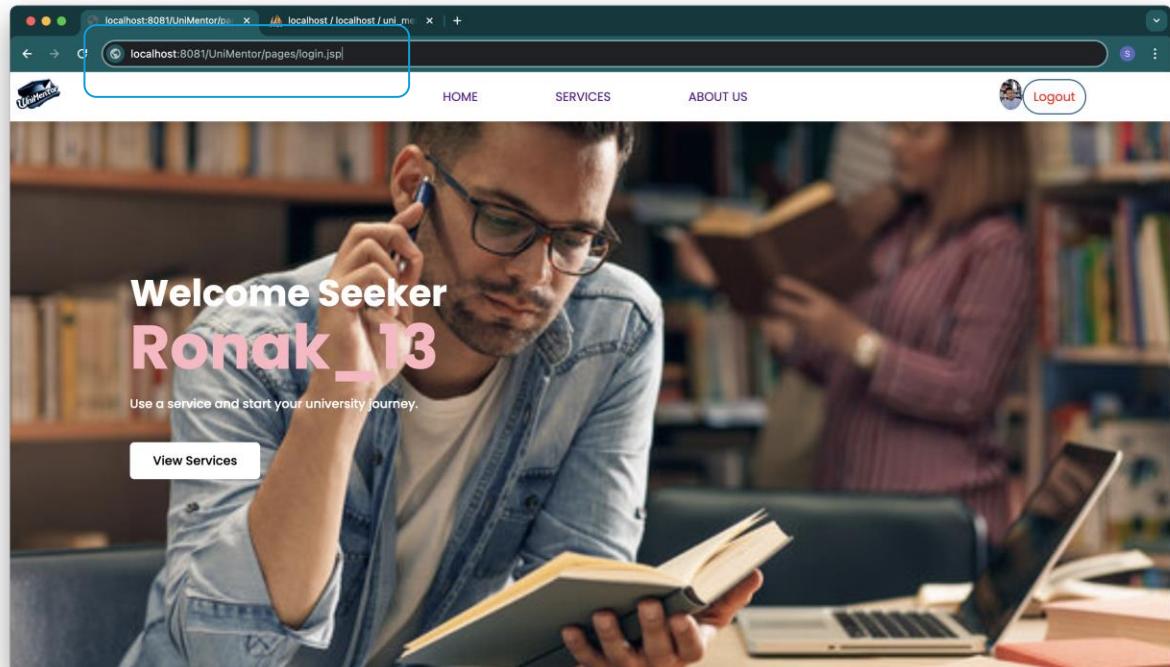


Figure 214: Screenshot of giving login page url

4) Welcome page displayed:

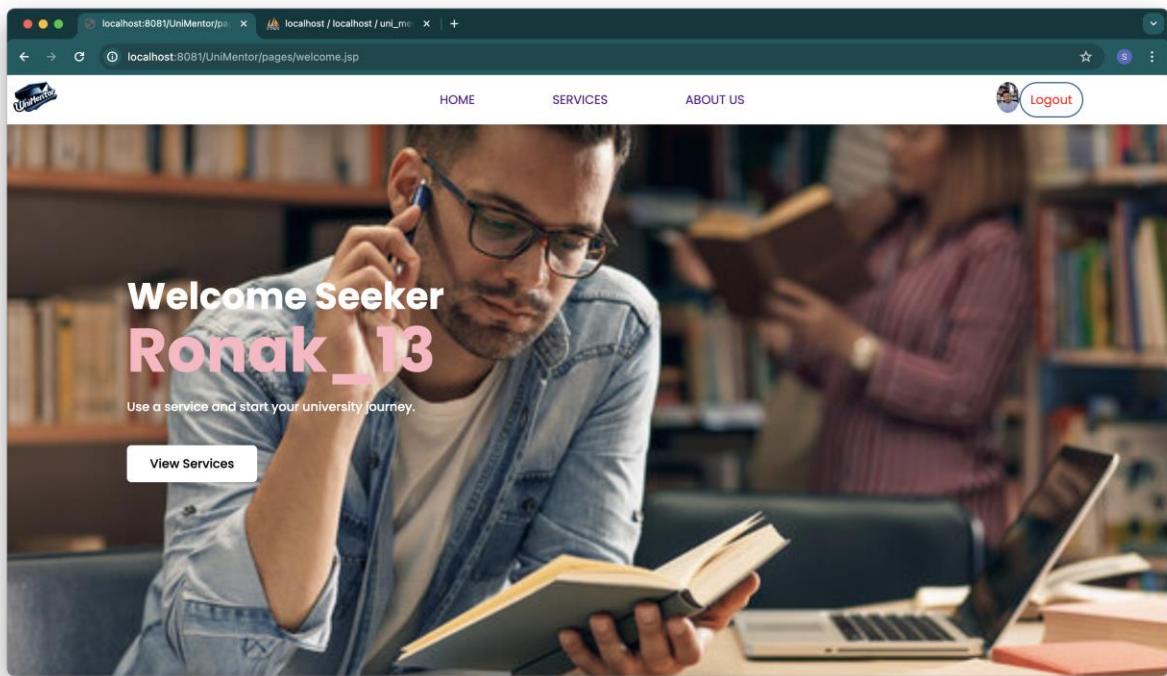


Figure 215: Screenshot of welcome page displayed

5) Giving register mentor page URL in address bar:

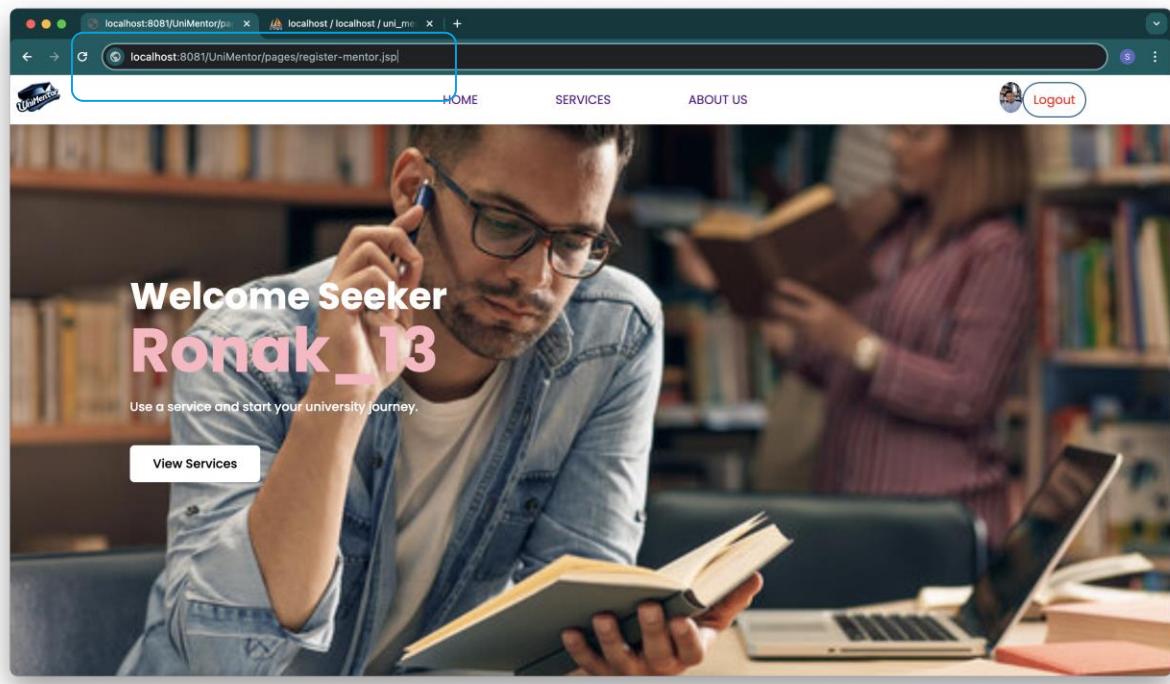


Figure 216: Screenshot of giving register mentor page url

6) Welcome page displayed:

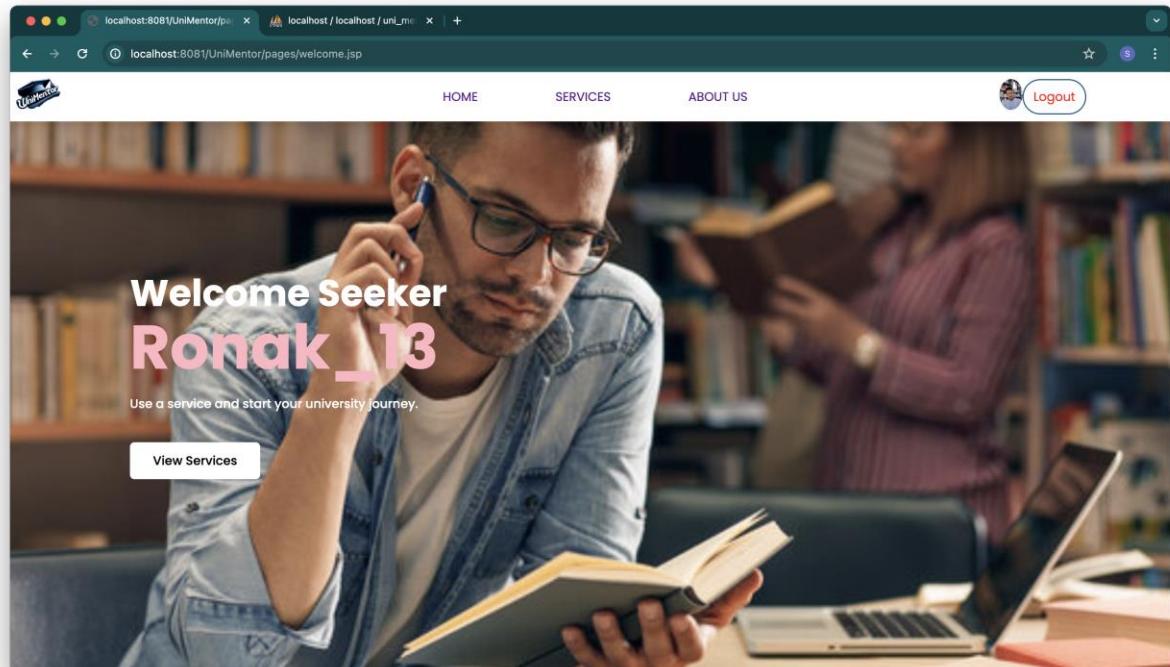


Figure 217: Screenshot of welcome page displayed

7) Giving register seeker page URL in address bar:

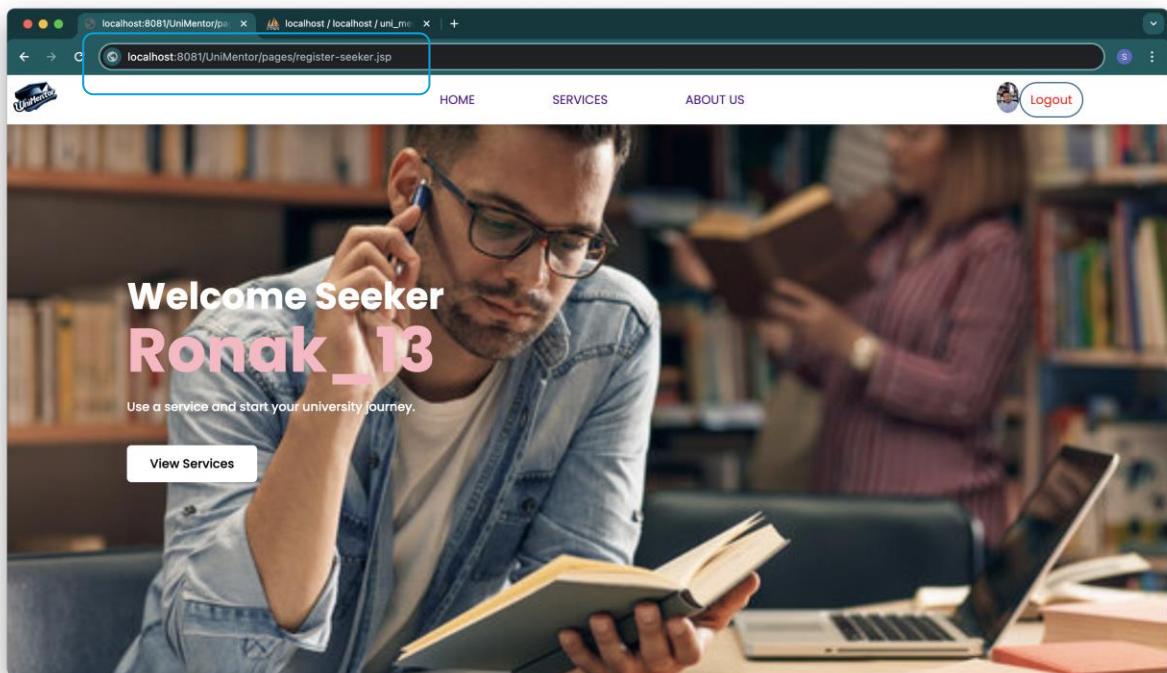


Figure 218: Screenshot of giving register seeker page url

8) Welcome page displayed:

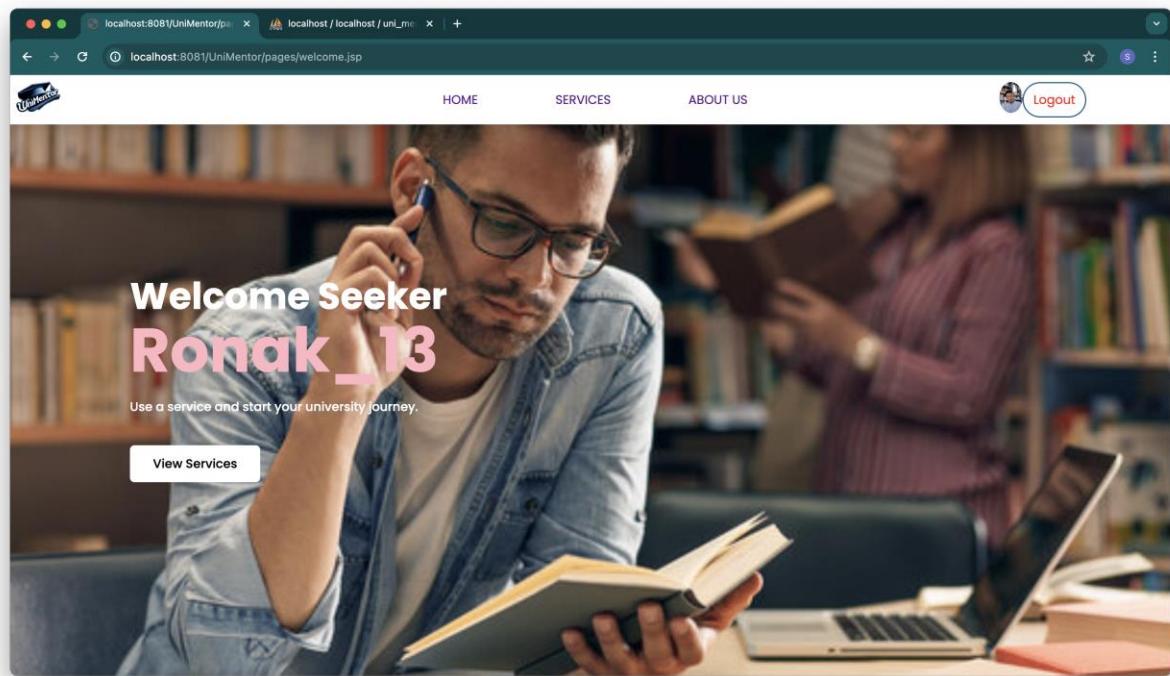


Figure 219: Screenshot of welcome page displayed

6. Tools and libraries used

3.1. Software Platform Selection

1) Eclipse

Eclipse is mainly known as an IDE for Java, also regarded as a Java development environment. However, it is far more than just a Java IDE due to its various features. It is a tools framework, an open-source enabler, a community, and an ecosystem (Eclipse Foundation AISBL, n.d.).



Figure 220: Logo of eclipse ide

Reason for selection:

- Used as a platform for system development.
- To integrate with Apache Tomcat. It helps to easily deploy, manage, and test servlets and JSP pages directly from the Eclipse IDE.
- For features like auto-completion, error checking and debugging the code.

1.1 Apache Tomcat

Apache Tomcat is a Java servlet container that is freely available and open-source. Its fundamental role is providing a runtime environment for Java web applications and websites. It has numerous other advantages as well. It is incredibly lightweight and fast (LogicMonitor Inc, 2024).

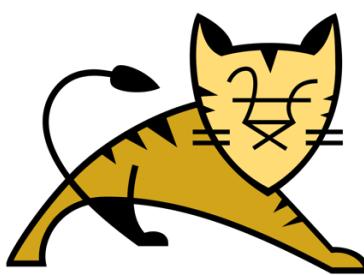


Figure 221: Logo of tomcat server

Reason for selection:

- To locally host and run servlets and JSP pages during development.
- To simulate the behavior of a production environment locally, enabling efficient debugging and validation of web applications.

2) XAMPP

XAMPP is widely recognized for PHP development environments. It's completely free and comes with Apache, MariaDB, PHP, and Perl in one package. It is designed for user-friendly installation and utilization (Apache Friends, 2023).



Figure 222: Logo of XAMPP

Reason for selection:

- The apache server of XAMPP hosts the website locally.
- The MySQL database provided by XAMPP will be used to store and manage data in the database.

3) Balsamiq

Balsamiq wireframes is a software designed to help user create digital sketch of applications and websites interface before any coding. This tool is used to design wireframes to serve communication and help user clarify his idea or obtain a raw sketch of the product that is to be made (balsamiq, 2023).



Figure 223: Logo of balsamiq

Reason for selection:

- To design the wireframes of the pages of the website.

4) MS Word

Microsoft released MS Word, a program for making documents, on October 25, 1983. It helps create professional stuff like letters and reports. With its many features, we can easily edit and format files and documents (BYJU'S, 2021).



Figure 224: Logo of MS Word

Reason for selection:

- To write the proposal and report of the program. It offers features like auto correction, fonts, report design, pdf conversion, and many more.

5) Draw.io

Draw.io is a widely used software for crafting charts and diagrams, offering a variety of shapes and decorative elements to add a unique look to our diagrams. The easy-to-use drag-and-drop functionality simplifies the process of creating graphs and charts, and the user interface is user-friendly. (Hope, Computer, 2020).



Figure 225: Logo of draw.io

Reason for selection:

- To draw and design entity relationship diagram (ERD) of the database after normalization.

3.2. Programming PS

1) Java

Java is a robust, object-oriented programming language with a class-based structure, designed for various applications. By compiling programs into bytecode, developers can write code once and run it on any platform that supports Java. Java was the most popular language in 2019. (Wikimedia Foundation, Inc, 2023).

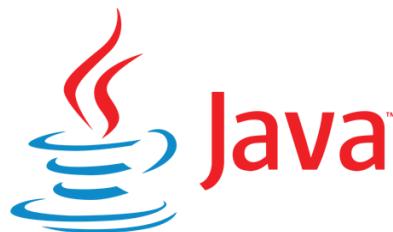


Figure 226: Logo of Java

Reason for selection:

- All the classes like controller, filter, servlets, models and utils will be made using Java.
- All the methods and functions of these classes are will be written in Java Programming Language.
- All the Java Server Pages(JSP) will also include java code along with html.

2) HTML, CSS, and JS

HTML, CSS, and JavaScript build on one another. They can form a simple website to a website with advanced interactive features. HTML sets up the basic structure of a website, CSS makes it look good, and JavaScript adds functionality to make it interactive (The University of Texas at Austin, 2021).



Figure 227: HTML, CSS AND JS LOGO

Reason for selection:

- HTML will be used to define the structure of the web pages.
- CSS will be used for visually appealing design and responsiveness.
- JS will be used for features like animation and other functions.

3) MySQL

MySQL most popular relational database management system. It is developed, distributed, and supported by Oracle Corporation. It is a database management system, open source, has fast and reliable server, and, works in client/server or embedded systems (Oracle, 2024).



Figure 228: Logo of MySQL

Reason for selection:

- To store and manage structured data for the website.

- It provides SQL (Structured Query Language) as its querying language, which allows to perform different operations.
- It provides mechanisms for enforcing data integrity constraints such as primary keys, foreign keys and so on.
- It will be suitable for handling large volume of data.

3.3. Framework and libraries

1) Java Servlets & Java Server Pages (JSPs)

They are Java-based programs that operate on a Java application server, enhancing the functionalities of a web server. Servlets are essentially Java classes crafted to handle HTTP requests within a web application. JSPs serve as an extension of HTML, allowing for the integration of Java code snippets directly into HTML pages. These Java segments generate dynamic content (IBM, 2023).

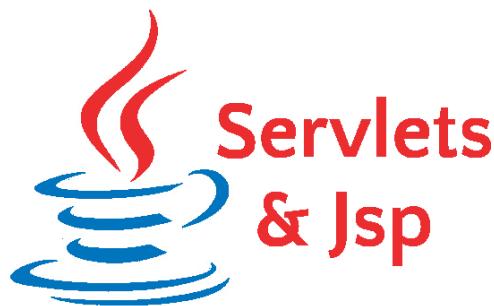


Figure 229: Figure of servlets and JSP

Reason for selection:

- Servlets are used to handle server-side logic like http request, database interaction, page navigation and making dynamic responses.
- JSP is used to create dynamic web pages by integrating java code in HTML.
- JSPs are also the user interface of the website.

2) Java Database Connectivity (JDBC) Driver

It provides a standard way to access data using the Java programming language. With JDBC, applications can interact with various databases and operate on any platform equipped with a Java Virtual Machine. It allows application to send SQL statements to different data sources (Progress Software Corporation, 2024).

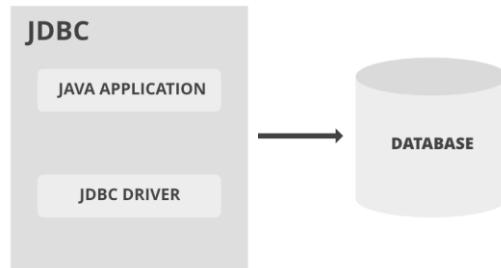


Figure 230: Diagram of JDBC

Reason for selection:

- JDBC driver is used to establish connection between Java web application and the MySQL database.
- It allows statement execution which allows to perform CRUD (Create, Read, Update, Delete) operations as needed.

7. Development Process

7.1. Creating a new dynamic web project

At the very first, our dynamic web project is created from the Eclipse IDE. It will contain all the jsp, css, java files where our system will be built.

7.1.1 Project name with location, Apache Tomcat v8.5 and Dynamic module version 3.1

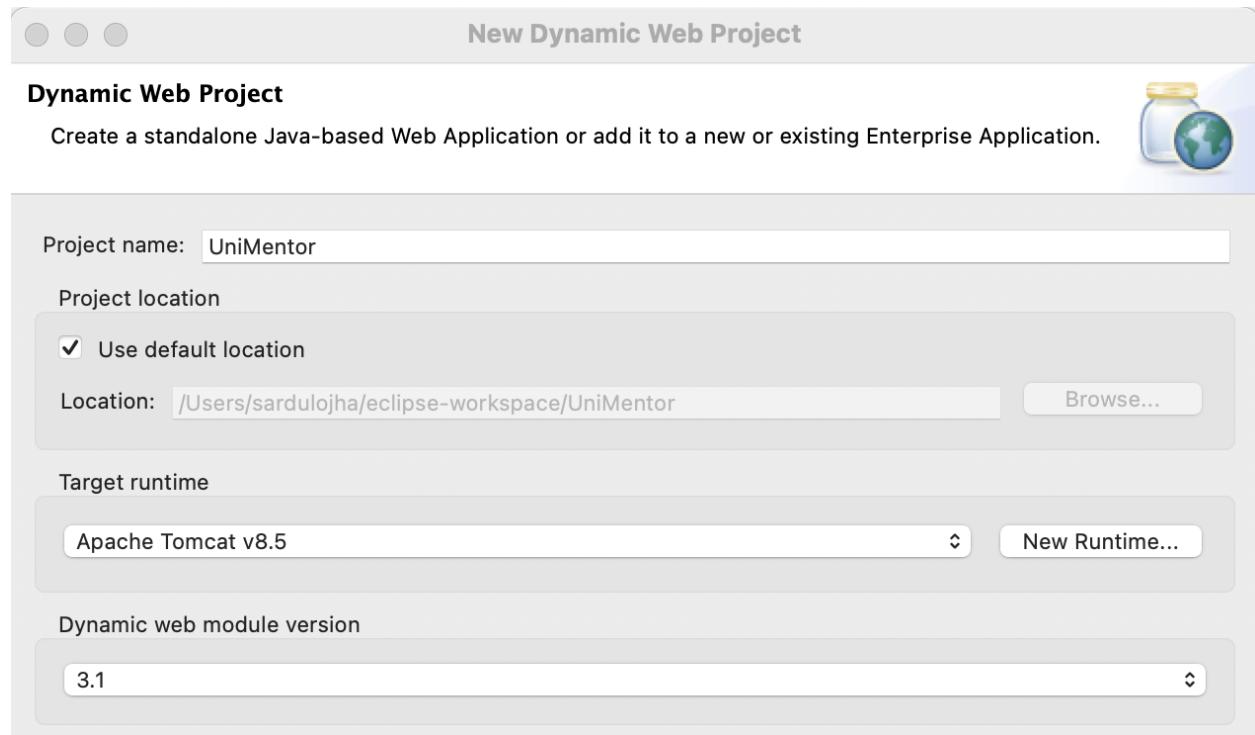


Figure 231: Screenshot of giving project name and selecting server

7.1.2. Selecting Generate web.xml deployment descriptor

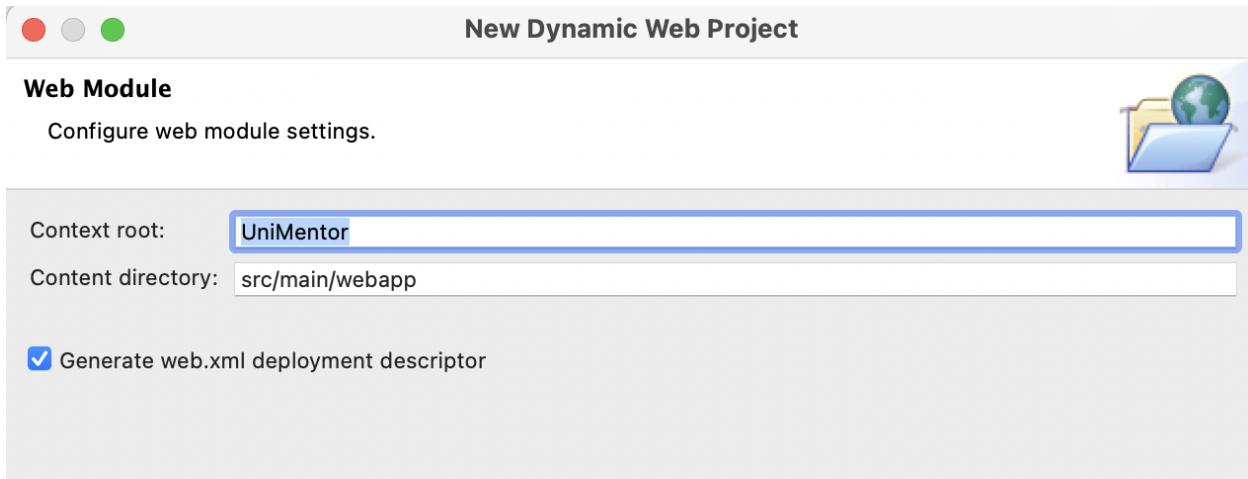


Figure 232: Screenshot of generating webxml description

7.1.3. Creating MVC pattern Packages and Folders

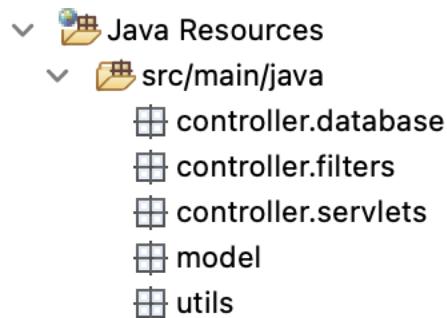


Figure 233: Screenshot of creating packages

7.1.4. Creating Folders inside webapp folder

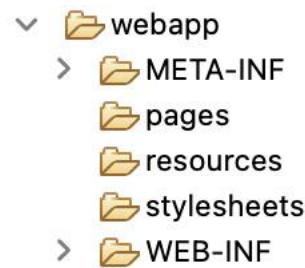


Figure 234: Screenshot of creating folders

7.1.5. Adding MySQL Connector jar file in deployment assembly

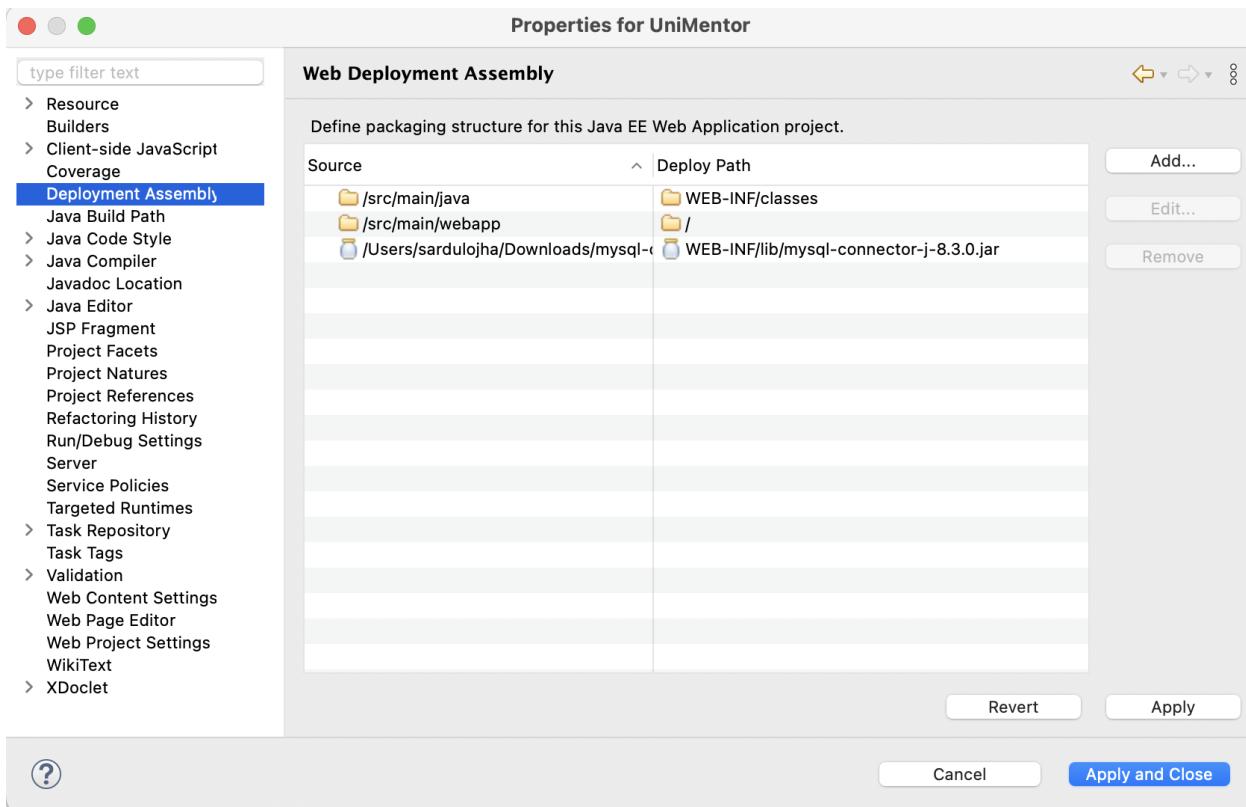


Figure 235: Screenshot of adding mysql jar in deployment assembly

7.1.6. Adding MySQL Connector jar file in java build path

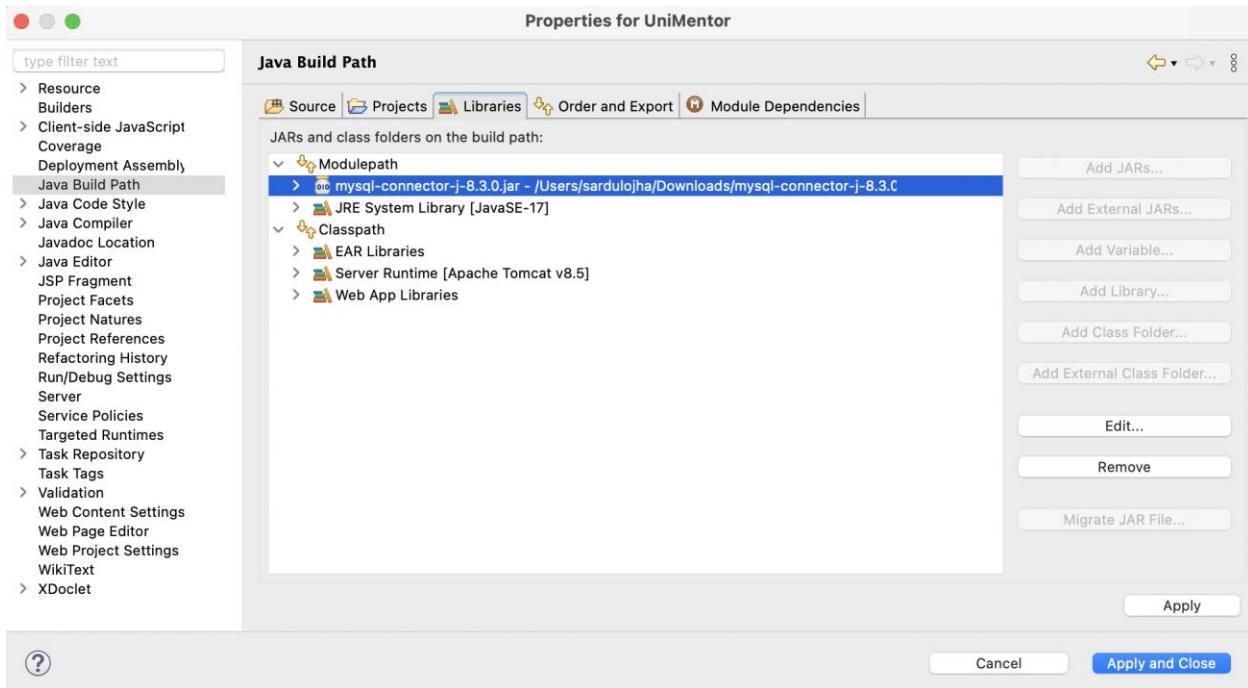


Figure 236: Screenshot of adding mysql jar in java build path

7.1.7. Creating DbController Class



Figure 237: Screenshot of creating DbController class

7.1.8. Creating AuthenticationFilter Class

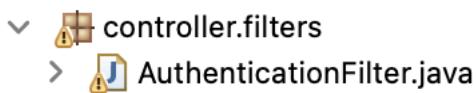


Figure 238: Screenshot of creating authentication filter class

7.1.9. Creating Servlet Classes

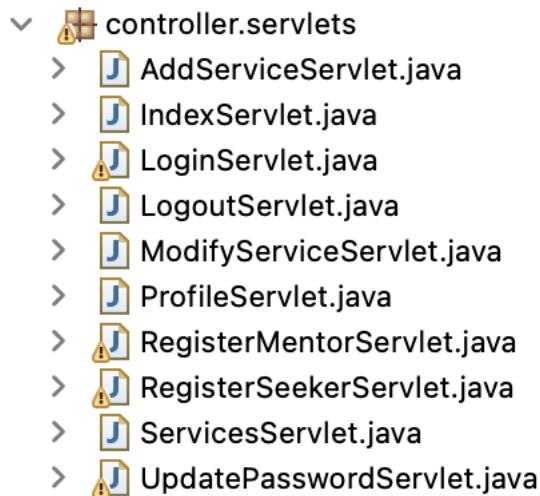


Figure 239: Screenshot of creating servlet classes

7.1.10. Creating Model Class



Figure 240: Screenshot of creating model class

7.1.11. Creating Util Class

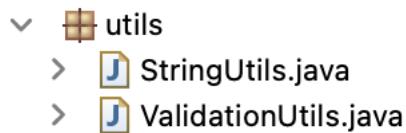


Figure 241: Screenshot of creating util classes

7.1.12. Creating JSPs

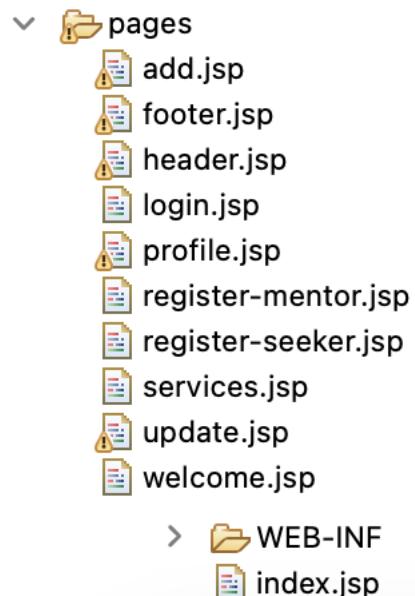


Figure 242: Screenshot of creating jsp

7.1.13. Creating Stylesheets (CSS)

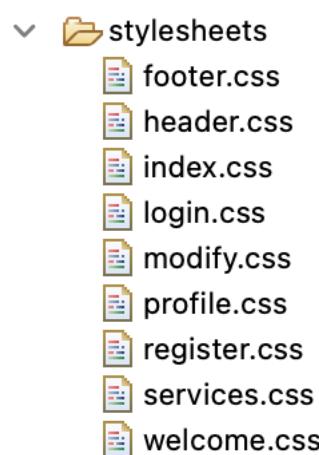


Figure 243: Screenshot of creating css files

7.2. Designing Wireframes

The second step was designing wireframes. I used balsamiq software to design the wireframe of the entire system. It was very important for planning out how the “UniMentor” would look and work. One by one I designed wireframes for all the 9 pages.

Designing wireframe in balsamiq:

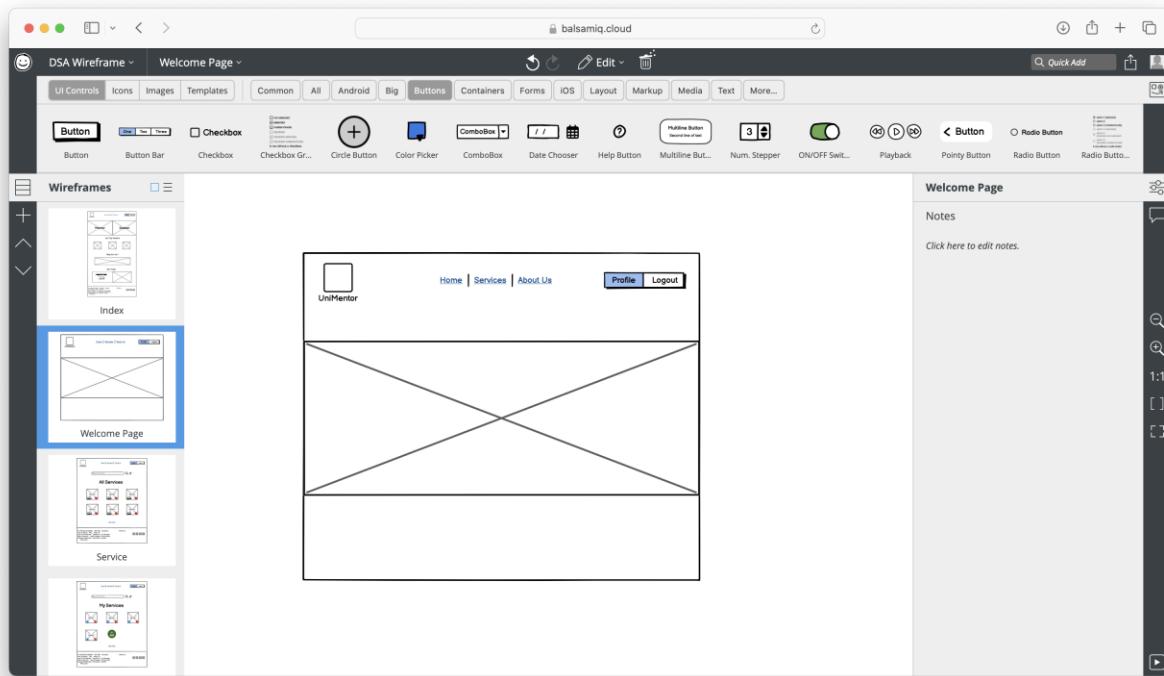


Figure 244: Screenshot of designing wireframes

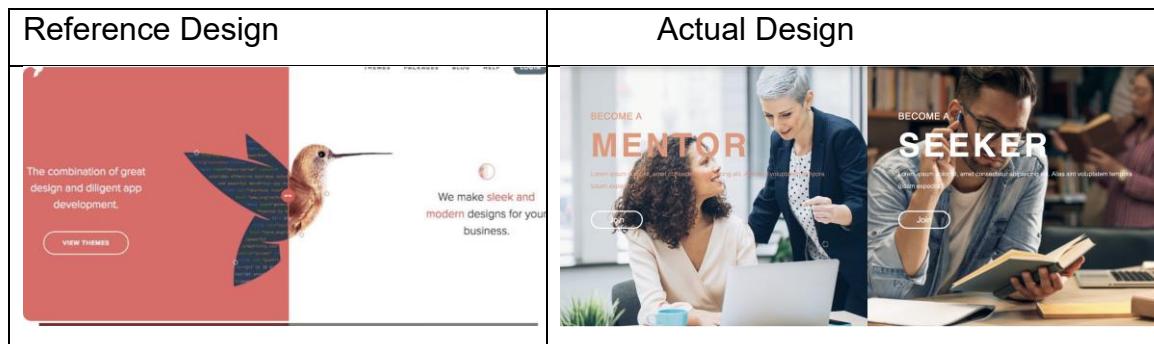
7.3. Website design (Frontend)

Once the wireframes were done, I jumped into building the actual web pages using HTML, CSS, and JavaScript. Each page, like the Home Page, Profile Page and Service Page, and, got special attention to make sure they looked good and were easy to use. For the Home Page, I made sure you could see a list of mentors. The Profile Page let users check and update their info, with colors and styles making it look nice, and some fancy coding making sure everything works smoothly. The Service Page made it easy to check service, with colors and sorting making it less confusing, and some clever coding making it all interactive. All along the way, I made sure everything was easy to understand and use for everyone, no matter what device they're using.

7.3.1. Home page

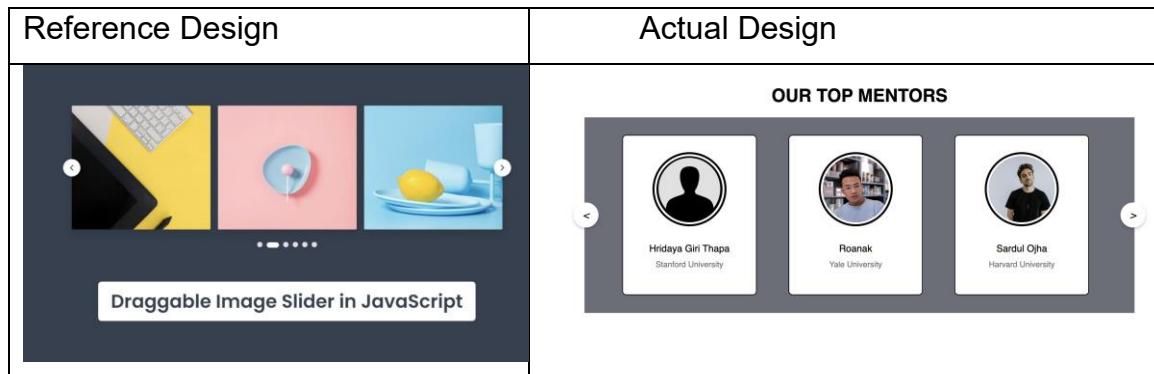
1) Split Screen

- Design Reference: <https://www.pinterest.com/pin/examples-of-beautiful-splitscreen-layoutsdesign-inspiration--66850375701714906/>
- Code Reference: <https://github.com/codict/Split-Landing-Page/blob/master/index.html>



2) Most Popular

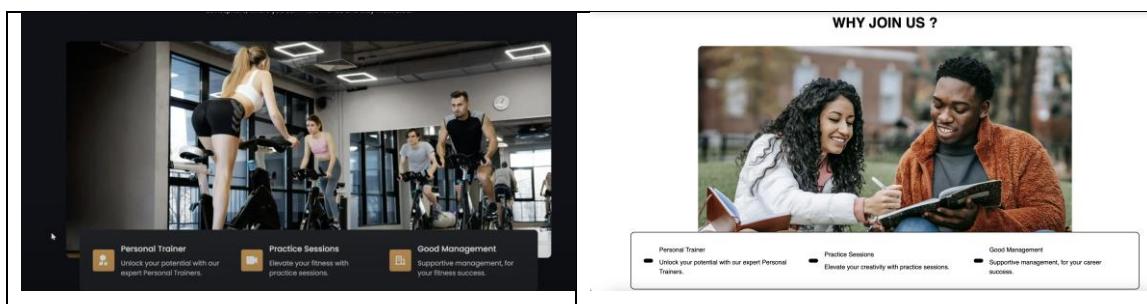
- Design Reference: <https://www.codingnepalweb.com/draggable-image-slider-html-css-javascript/>
- Code Reference: <https://www.codingnepalweb.com/draggable-card-slider-html-css-javascript/>



3) Why Join Us

- Design Reference: https://www.youtube.com/watch?v=_rWKTxvTUzQ
- Code Reference: https://www.youtube.com/watch?v=_rWKTxvTUzQ





4) Join

- Design Reference: <https://www.fiverr.com>
- Code Reference: <https://www.youtube.com/watch?v=rzP0BpDpXY4&t=67s>

Reference Design	Actual Design

7.3.2. Service page

1) Search Bar

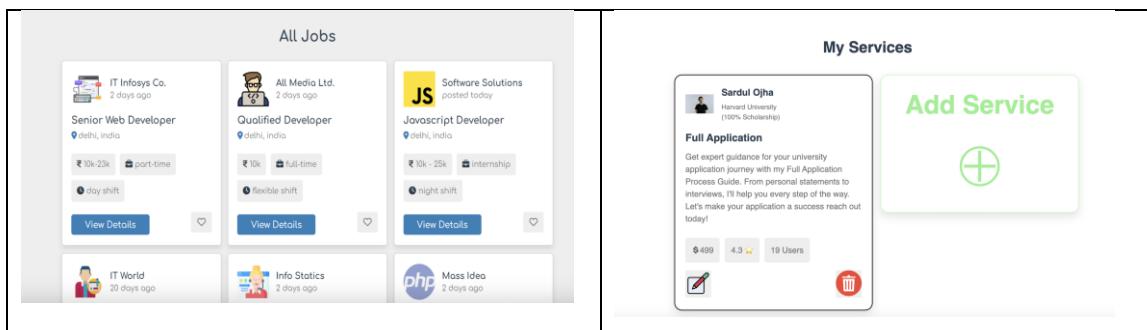
- Design Reference: <https://stackoverflow.com/questions/62975335/place-the-dropdown-list-inside-the-search-bar-using-html>
- Code Reference: <https://www.patreon.com/posts/78939892?pr=true>

Reference Design	Actual Design

2) Service List

- Design Reference:
https://github.com/umeshkillua/Jobs_Portal/blob/main/screenshots/all%20obs.png
- Code Reference: https://github.com/umesh-killua/Jobs_Portal/blob/main/screenshots/all%20jobs.png

Reference Design	Actual Design



7.3.3. Profile page

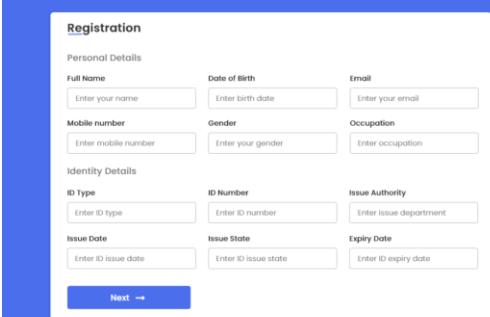
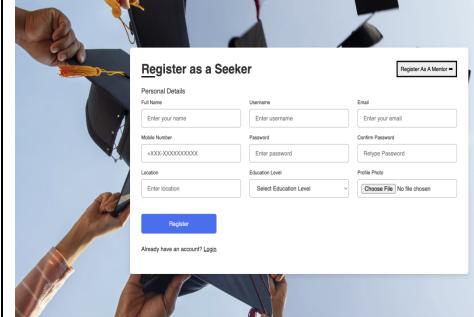
- Design Reference: <https://codepen.io/doantthuylinh98/pen/OJOBKwe>
- Code Reference: <https://codepen.io/doantthuylinh98/pen/OJOBKwe>

Reference Design	Actual Design																						
<p>Registration</p> <table> <tr> <td>Full Name <input type="text" value="E.g. John Smith"/></td> <td>Username <input type="text" value="johnWC98"/></td> </tr> <tr> <td>Email <input type="text" value="johnsmith@hotmail.com"/></td> <td>Phone Number <input type="text" value="012-345-6789"/></td> </tr> <tr> <td>Password <input type="password" value="*****"/></td> <td>Confirm Password <input type="password" value="*****"/></td> </tr> <tr> <td colspan="2"> Gender <input type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Prefer not to say </td> </tr> <tr> <td colspan="2" style="text-align: center;">Register</td> </tr> </table>	Full Name <input type="text" value="E.g. John Smith"/>	Username <input type="text" value="johnWC98"/>	Email <input type="text" value="johnsmith@hotmail.com"/>	Phone Number <input type="text" value="012-345-6789"/>	Password <input type="password" value="*****"/>	Confirm Password <input type="password" value="*****"/>	Gender <input type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Prefer not to say		Register		<p>Your Profile</p> <table> <tr> <td>Full Name <input type="text" value="Sardul Ojha"/></td> <td>Username <input type="text" value="Sardul_11"/></td> </tr> <tr> <td>Email <input type="text" value="sardul@gmail.com"/></td> <td>Phone Number <input type="text" value="+977-9742855801"/></td> </tr> <tr> <td colspan="2" style="text-align: center;">Update Profile</td> </tr> <tr> <td>Old Password <input type="password" value="Enter previous password"/></td> <td>New Password <input type="password" value="Enter new password"/></td> </tr> <tr> <td colspan="2" style="text-align: center;">Update Password</td> </tr> <tr> <td colspan="2" style="text-align: center;">Cancel Delete Account</td> </tr> </table>	Full Name <input type="text" value="Sardul Ojha"/>	Username <input type="text" value="Sardul_11"/>	Email <input type="text" value="sardul@gmail.com"/>	Phone Number <input type="text" value="+977-9742855801"/>	Update Profile		Old Password <input type="password" value="Enter previous password"/>	New Password <input type="password" value="Enter new password"/>	Update Password		Cancel Delete Account	
Full Name <input type="text" value="E.g. John Smith"/>	Username <input type="text" value="johnWC98"/>																						
Email <input type="text" value="johnsmith@hotmail.com"/>	Phone Number <input type="text" value="012-345-6789"/>																						
Password <input type="password" value="*****"/>	Confirm Password <input type="password" value="*****"/>																						
Gender <input type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Prefer not to say																							
Register																							
Full Name <input type="text" value="Sardul Ojha"/>	Username <input type="text" value="Sardul_11"/>																						
Email <input type="text" value="sardul@gmail.com"/>	Phone Number <input type="text" value="+977-9742855801"/>																						
Update Profile																							
Old Password <input type="password" value="Enter previous password"/>	New Password <input type="password" value="Enter new password"/>																						
Update Password																							
Cancel Delete Account																							

7.3.4. Register page

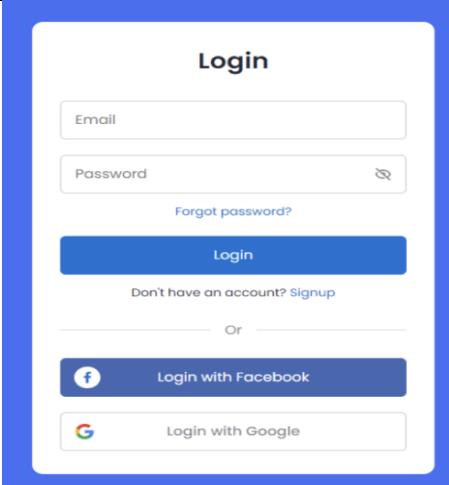
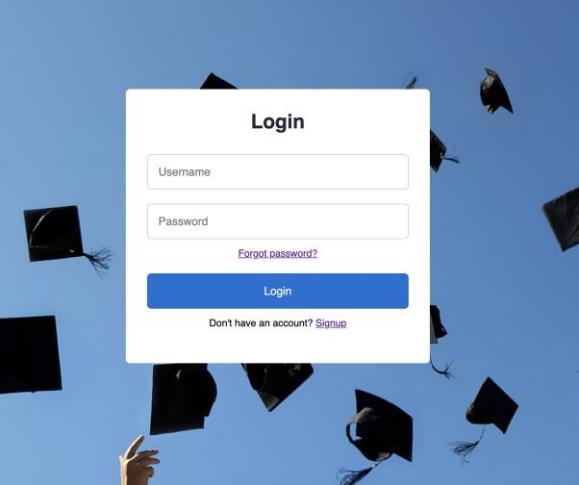
- Design Reference: <https://www.codingnepalweb.com/registration-form-html-css-javascript/>
- Code Reference: <https://www.codingnepalweb.com/registration-form-html-css-javascript/>

Reference Design	Actual Design
------------------	---------------

 <p>The registration form has a blue header with the title 'Registration'. It contains three main sections: 'Personal Details' (Full Name, Date of Birth, Email, Mobile number, Gender, Occupation), 'Identity Details' (ID Type, ID Number, Issue Authority, Issue Date, Issue State, Expiry Date), and a 'Next →' button.</p>	 <p>The seeker registration form is overlaid on a background image of people throwing graduation caps. It includes fields for Personal Details (Full Name, Username, Email, Mobile Number, Password, Confirm Password, Location, Education Level, Profile Photo), a file upload field ('Choose File'), and a 'Register' button.</p>
--	---

7.3.5. Login page

- Design Reference: <https://www.codingnepalweb.com/login-signup-form-html-css-javascript/>
- Code Reference: <https://www.codingnepalweb.com/login-signup-form-html-css-javascript/>

Reference Design	Actual Design
 <p>The login form has a white header with the title 'Login'. It includes fields for 'Email' and 'Password', a 'Forgot password?' link, a large blue 'Login' button, and social media login options for 'Login with Facebook' and 'Login with Google'.</p>	 <p>The login form is centered over a background image of many graduation caps flying in the air. It has fields for 'Username' and 'Password', a 'Forgot password?' link, a blue 'Login' button, and a 'Don't have an account? Signup' link.</p>

7.3.6. Add/Update page

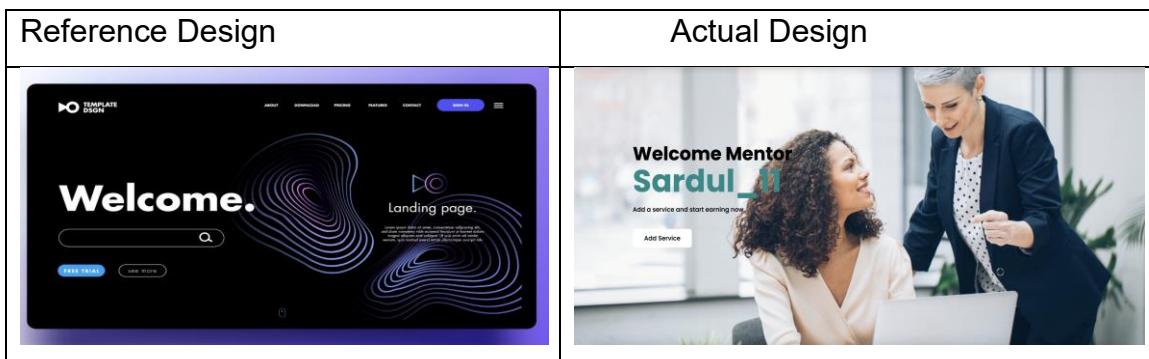
- Design Reference: <https://codepen.io/peterfaretra/pen/oNjRzje>
- Code Reference: <https://codepen.io/peterfaretra/pen/oNjRzje>

Reference Design	Actual Design
------------------	---------------

<p>NAME *</p> <input type="text"/> <p>EMAIL *</p> <input type="text"/> <p>MESSAGE</p> <input type="text"/> <p>Submit</p>	<p>Add Service</p> <p>TITLE</p> <input type="text"/> <p>DESCRIPTION</p> <input type="text"/> <p>PRICE</p> <input type="text"/> <p>Add</p>
--	--

7.3.7. Welcome page

- Design Reference: <https://www.searchenginejournal.com/best-landing-pages/494196/>



7.3.8. Header

- Design Reference: <https://codepen.io/hitensharma/pen/dybryGE>
- Code Reference: <https://codepen.io/hitensharma/pen/dybryGE>



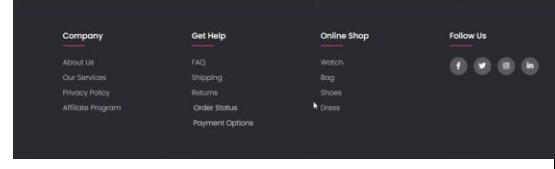
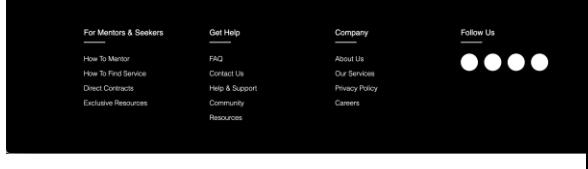
7.3.9. Footer

- Design Reference:

<https://www.youtube.com/watch?app=desktop&v=YOb67OKw62s>

- Code Reference:

<https://www.youtube.com/watch?app=desktop&v=YOb67OKw62s>

Reference Design	Actual Design
 <p>The reference design footer is dark-themed with white text. It has four main columns: 'Company' (About Us, Our Services, Privacy Policy, Affiliate Program), 'Get Help' (FAQ, Shipping, Returns, Order Status, Payment Options), 'Online Shop' (Watch, Bag, Shoes, Dress), and 'Follow Us' (Facebook, Twitter, Instagram, LinkedIn). Below these are three small circular icons.</p>	 <p>The actual design footer is also dark-themed with white text. It has four main columns: 'For Mentors & Seekers' (How To Mentor, How To Find Service, Direct Contracts, Exclusive Resources), 'Get Help' (FAQ, Contact Us, Help & Support, Community Resources), 'Company' (About Us, Our Services, Privacy Policy, Careers), and 'Follow Us' (four white circles). Below these are three small circular icons.</p>

7.4. Backend design

For the backend, I outlined the structure and functionality of the server-side components that power the “UniMentor” web application. Leveraging Java EE technologies such as Servlets and Java Database Connectivity (JDBC), I established a robust backend architecture capable of handling user authentication, data retrieval, and database operations. The database design, depicted through an Entity-Relationship (ER) diagram, efficiently organized mentor, seeker and service information ensuring data integrity and accessibility. Security measures, including user authentication and input validation, were implemented to safeguard sensitive data and prevent unauthorized access. Session management techniques were employed to track user sessions securely across multiple interactions with the application. Additionally, the backend functionality encompassed CRUD operations for managing mentor’s service. Scalability and performance considerations were addressed through optimized database queries and infrastructure planning, ensuring the backend could accommodate future growth and increased user demand. Similarly, CRUD for profile including image were also done.



Figure 245: Screenshot of backend development

7.5. Report

I summed up our entire project in one place. I started with the wireframes we made using Balsamiq, which helped us plan how each page would look. Then, I explained the Java classes we used to make the backend work, showing what each part of our code does. We also talked about the database tables we set up to store student info and attendance records, and I included a picture to show how they're connected. After that, I listed the different tests we did to make sure everything works as it should. We also talked about the tools we used throughout the project and why they were important. Lastly, I looked at what went well, what were the challenges and errors, and what we could improve on in the future. This report gives a clear picture of our project journey and how we built the "UniMentor" step by step.

8. Critical Analysis

8.1. Challenges

There were many challenges that I had to overcome during my journey of building this program. Two of the most difficult challenges to overcome, for me, are listed below:

8.1.1. Challenge 1: Displaying information from two different tables (i.e. service and mentor tables) in the service page

The first challenging part of this for me was to display mentor's information in his/her created service. When I consulted my teacher about this, he told me to use join query to retrive information from the database. He also advised me to create a separate mentor-service model (new model) to handle the data retrived and display in the service page. But I did not want to create separate model for this small task.

Upon deep thinking, research and evaluation, I realized about a data structure that can hold key value pairs i.e. HashMap. I revised our previous lecture slides about hashmap and also watched a short youtube video ([click for link](#)) about HashMaps. After a good knowledge and understanding about this topic, I implemented them in the code as shown below:

```

eclipse-workspace - UniMentor/src/main/java/controller/controller/ServicesServlet.java - Eclipse IDE

31 * @param request  The HttpServletRequest object
32 * @param response The HttpServletResponse object
33 * @throws ServletException if an exception occurs
34 * @throws IOException   if an I/O error occurs
35 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
36 */
37 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
38
39     // Getting attributes from current session
40     HttpSession userSession = request.getSession();
41     String currentUser = (String) userSession.getAttribute("username");
42     String currentUserType = (String) userSession.getAttribute("type");
43
44     // Extract search information from register page request parameters
45     String searchKey = request.getParameter("search");
46
47     // FOR SEEKER
48     if (currentUserType.equals("seeker")) {
49
50         // Declaring services arraylist to store all services
51         ArrayList<ServiceModel> services;
52
53         // Get searched services if search button clicked
54         if (searchKey == null) {
55             services = controller.getAllServiceInfo();
56         }
57
58         // Get all services for default
59         else {
60             services = controller.getSearchServiceInfo(searchKey);
61         }
62
63         // adding arraylist of services in request attribute to display in services jsp
64         request.setAttribute(StringUtil.SERVICE_LISTS, services);
65         request.setAttribute("isSeeker", true);
66
67
68         // Initializing and declaring hashMap to connect mentor details with their service
69         HashMap<String, MentorModel> hashMap = new HashMap<>();
70
71         // Loop on each items on arraylist
72         for (ServiceModel service : services) {
73             // Add mentor's name as key & its model as value
74             String currUsername = service.getMentorUsername();
75             MentorModel mentor = controller.getMentorInfo(currUsername);
76             hashMap.put(service.getMentorUsername(), mentor);
77         }
78
79         // adding the mentor hashMap in request attribute to display in services page
80         request.setAttribute("mentorMap", hashMap);
81
82         // FOR MENTOR
83     else {
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
337
338
339
339
340
341
342
343
344
345
345
346
347
347
348
349
349
350
351
352
353
353
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
139
```

Now the main challenge was to display the hashmap information in the service page(jsp). As you can see in the screenshot above the created hashmap was set as request attribute to send to the jsp. I did not know how to display the hashmap key and values so I googled for solution. I landed on stackoverflow(most popular developers community) and got this :

The screenshot shows a Stack Overflow question page. The question is titled "Thanks in advance." and has tags: java, javascript, jsp, jstl. It was asked by Braj on May 27, 2014, at 13:16, and last edited on the same day at 13:43. The question asks for help with displaying a map in JSP. A comment from Sachin follows, asking if the code is correct. Below the question, there are two answers. The first answer, which has 6 upvotes, provides a JSTL code snippet:

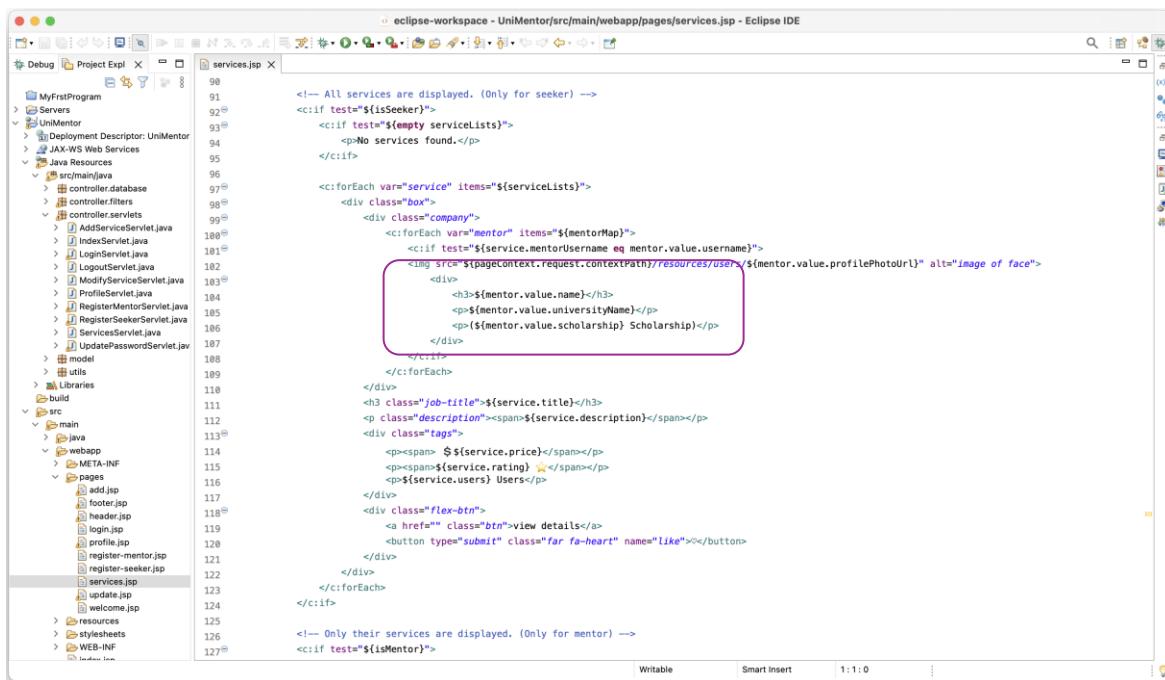
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach items="${currentLoggedInUsersMap}" var="entry">
    Key = ${entry.key}, value = ${entry.value}<br>
</c:forEach>
```

The second answer explains that it's similar to a Map.Entry in Java:

```
for (Map.Entry<String, String> entry : currentLoggedInUsersMap.entrySet()) {
    String key = entry.getKey();
    String value = entry.getValue();
}
```

Figure 247: Screenshot of getting solution from stack overflow

Upon using this expression language "\${hashMapObject.key}" and "\${hashMapObject.value}" I was able to retrieve information and display in the services page. The usage is shown below :



```

<!-- All services are displayed. (Only for seeker) -->
<c:if test="${!isSeeker}">
  <c:if test="${empty serviceLists}">
    <p>No services found.</p>
  </c:if>

  <c:forEach var="service" items="${serviceLists}">
    <div class="box">
      <div class="company">
        <c:forEach var="mentor" items="${mentorMap}">
          <c:if test="${service.mentorUsername eq mentor.value.username}">
            
          </c:if>
        </c:forEach>
        <h3>${service.title}</h3>
        <p class="description"><span>${service.description}</span></p>
        <div class="tags">
          <p><span> ${service.price}</span></p>
          <p><span>${service.rating} ⚡</span></p>
          <p>${service.users} Users</p>
        </div>
        <div class="flex-btn">
          <a href="#" class="btn">view details</a>
          <button type="submit" class="far fa-heart" name="like"></button>
        </div>
      </div>
    </c:forEach>
  </c:if>

  <!-- Only their services are displayed. (Only for mentor) -->
<c:if test="${isMentor}">

```

Figure 248: Screenshot of getting hashmap values in jsp

After this, the mentors and service details both were displayed in the service page. Hence, I tackled the challenge.

8.1.2. Challenge 2: Updating/Deleting related table elements from the database

Another challenge I faced was when updating/deleting the parent or child table rows from the database. Here, service and mentor table are related as primary key of mentor table is foreign key in service table. So, when I wanted to delete or mentor, it would throw an error as it was a in the parent table.

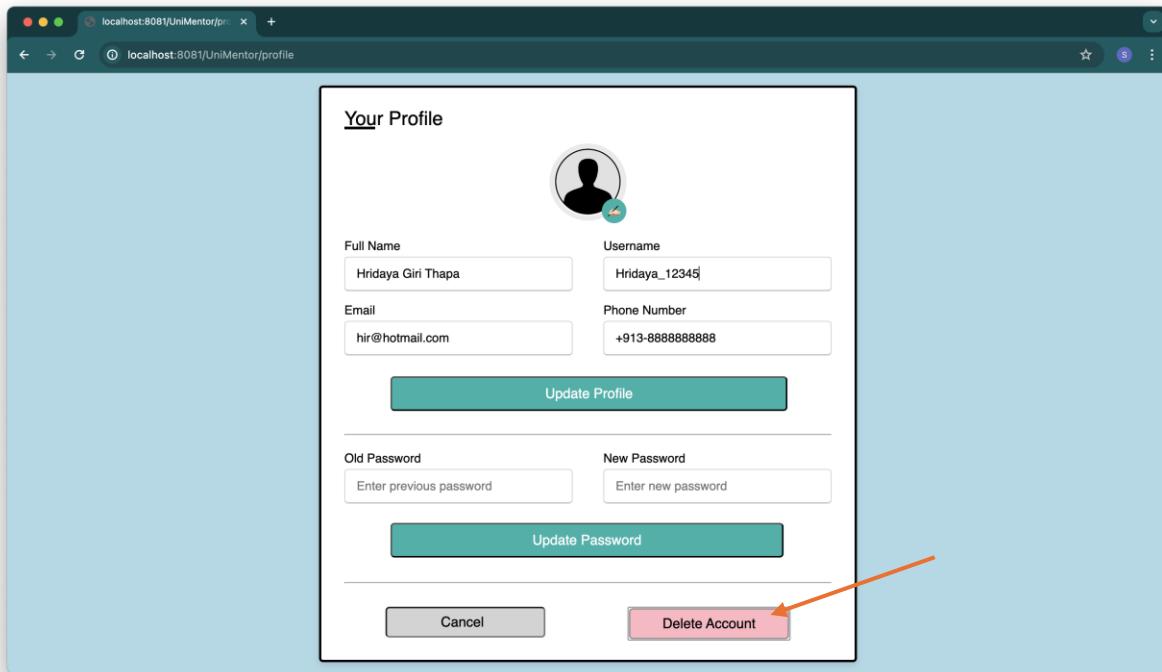


Figure 249: Screenshot of when delete button clicked

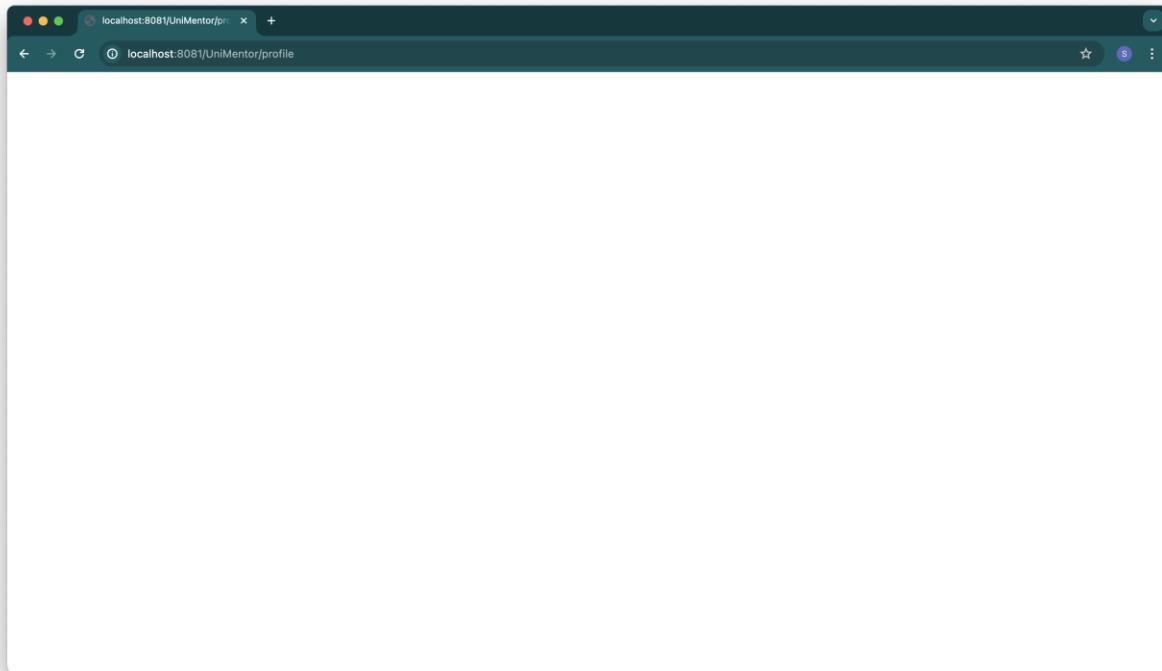


Figure 250: Screenshot of blank page displayed

The error was displayed as :

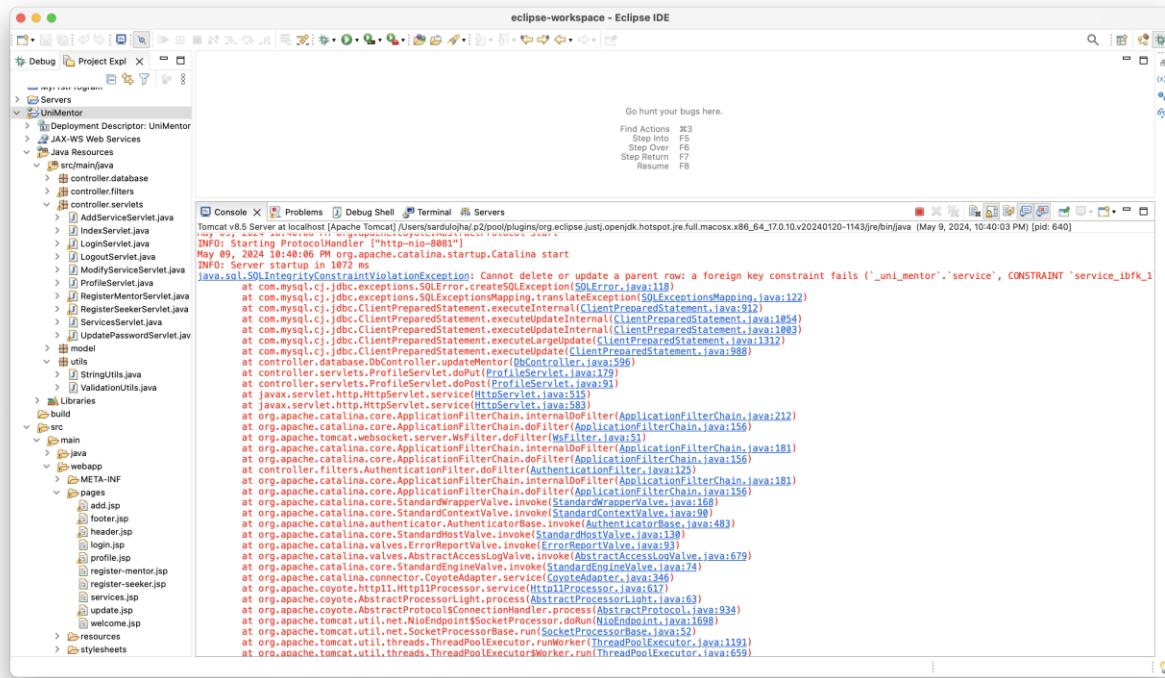


Figure 251: Screenshot of displayed error

When I tried updating the username/ deleting the user manually it would not let me do it.

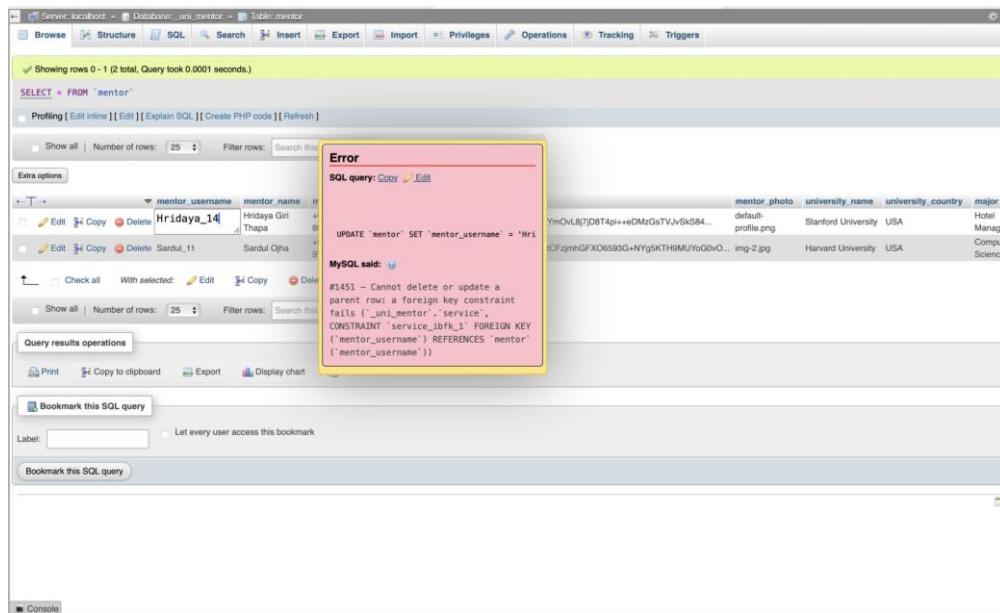


Figure 252: Screenshot of updating/deleting manually from database

Upon lots of research on internet, I came across this youtube video ([click for link](#)) which made me understand about “On Delete Cascade and Restrict”. I came to know that “on delete cascade” deletes or updates all the child relations whereas “on delete restrict”

restricts the deletion or update if there is a child relation. So, I changed the key relation to cascade in the place where I established the primary and foreign key relations.

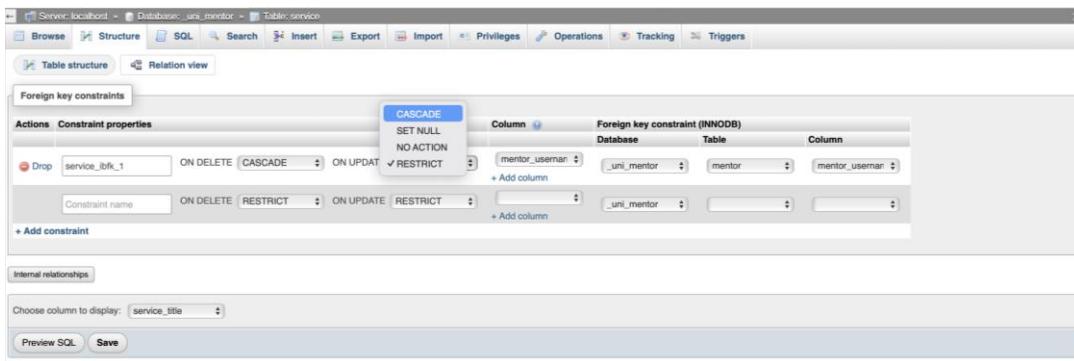


Figure 253: Screenshot of changing key type to cascade

Finally, the username was successfully updated and I was able to overcome this challenge.

8.2. Problems Faced

8.2.1. Problem Faced 1 : Tomcat Failed to start

Description : When I was working on my project, all of a sudden my project did not start. The reason was the tomcat server encountered a problem and failed to start. When I clicked on details to find out more about the problem, it read “Server Tomcat v8.5 Server at localhost failed to start”.

Screenshots of Problem:

1. Tomcat failed to start

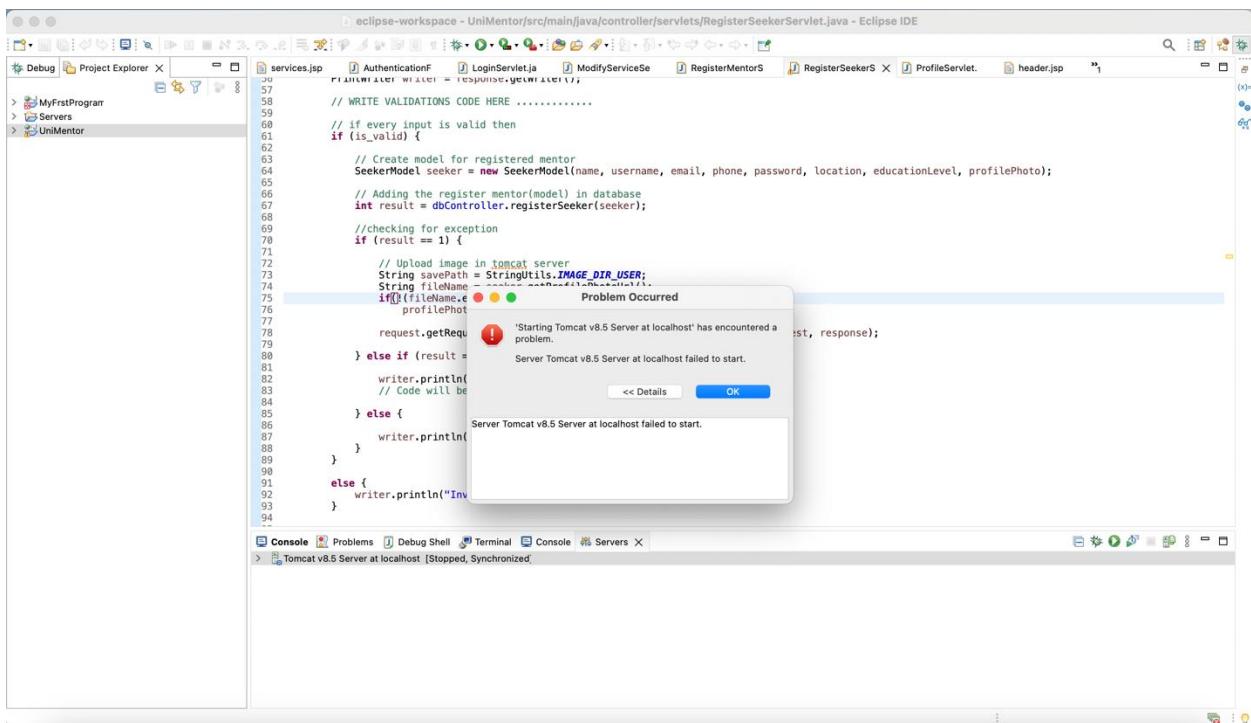


Figure 254: Screenshot of tomcat failed to start

Solution: I tried different methods to solve this problem. First I tried cleaning and restarting the server. But it did not work. I also checked for the same URL mapping of the servlets but it was't the case. I asked a friend for help and he suggested me to delete my previous programs. So, I deleted rest of the programs as they were of no use. Finally when I started the program, the tomcat started working and the problem I faced was solved.

Screenshots of attempt and solution:

1. Cleaning tomcat server

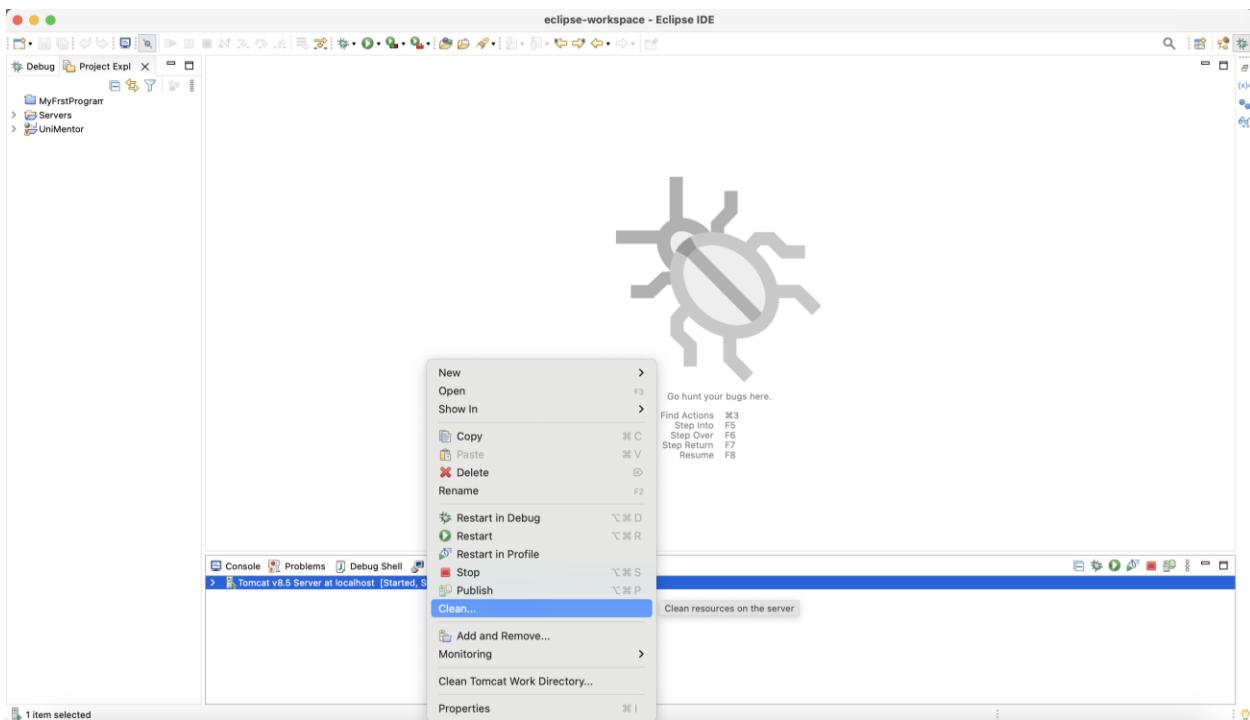


Figure 255: Screenshot of cleaning tomcat

2. Restarting tomcat server

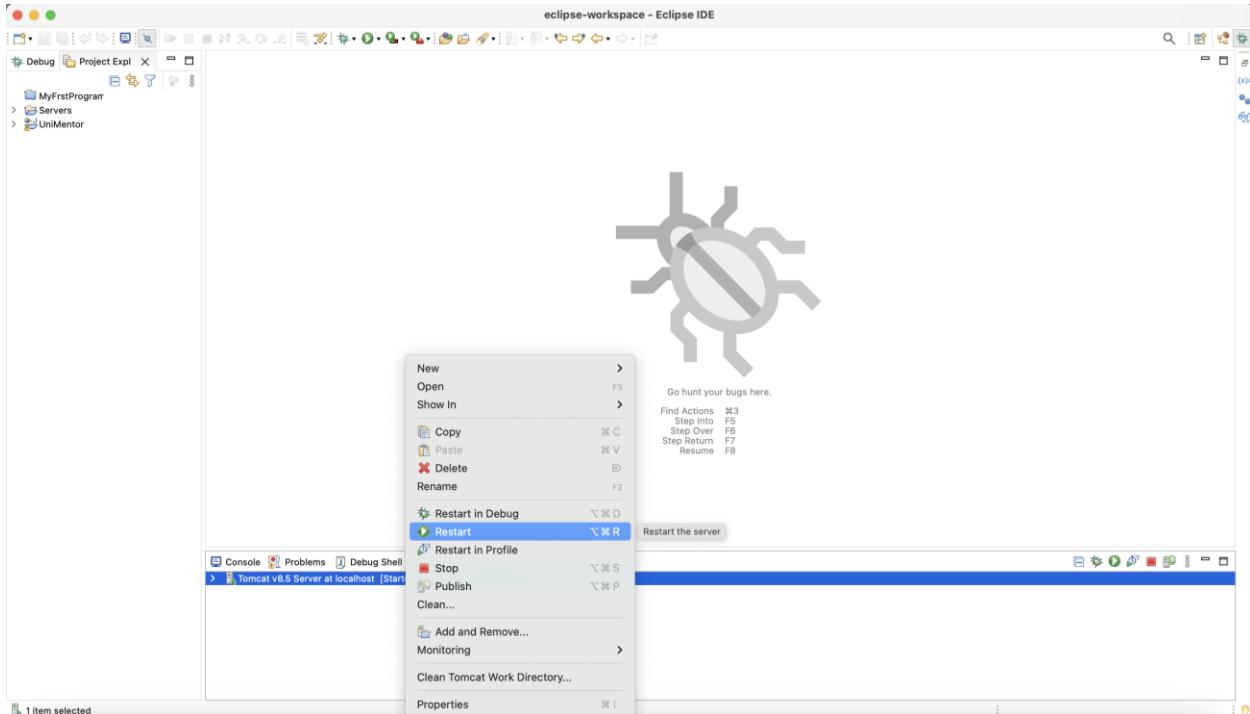


Figure 256: Screenshot of restarting tomcat

3. Deleting other projects (MyFirstProgram)

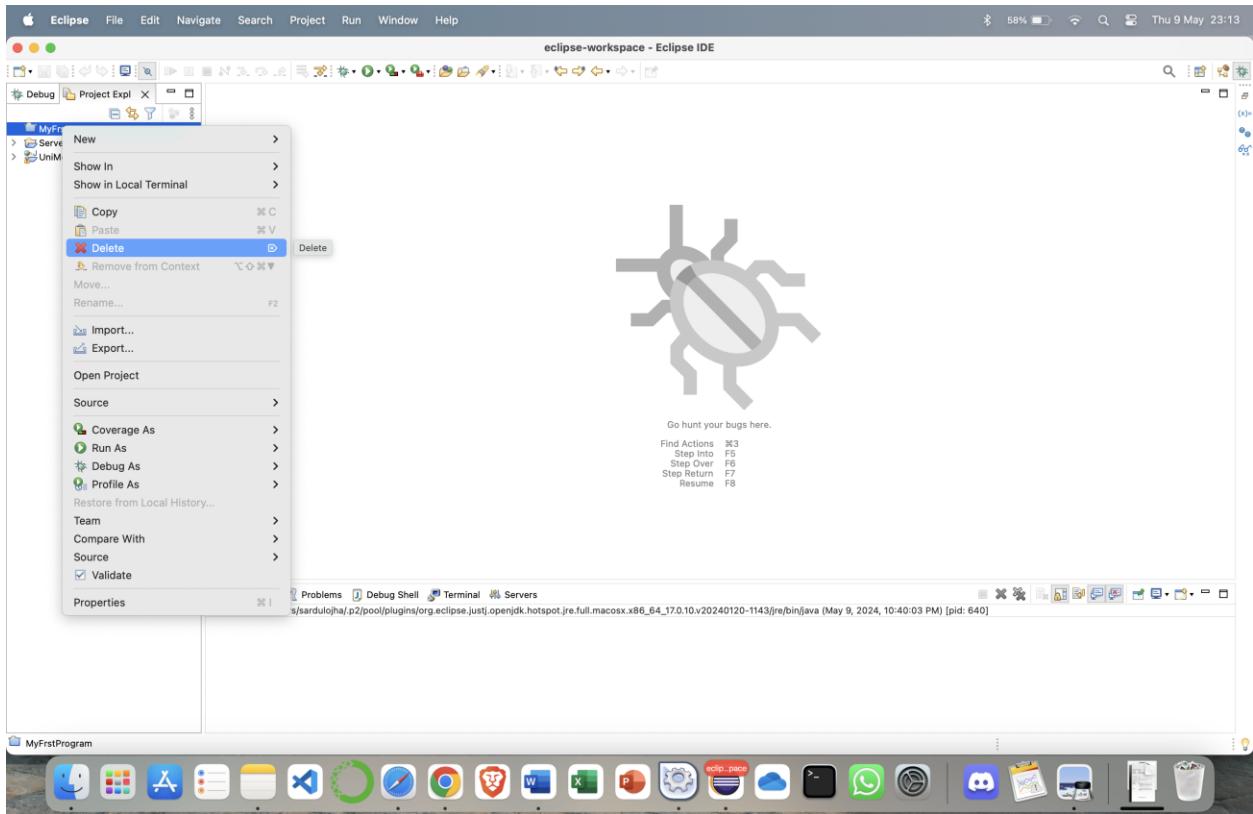


Figure 257: Screenshot of deleting other projects

4. Running the program

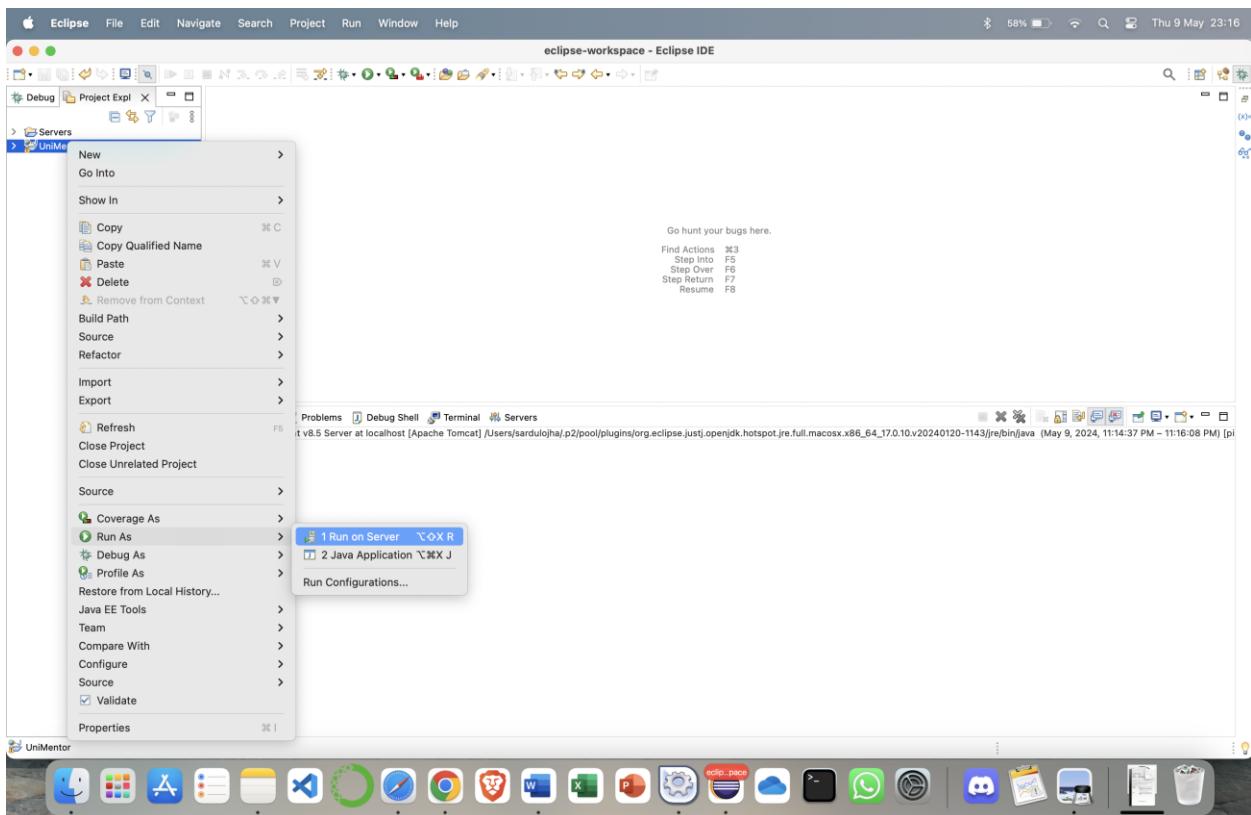


Figure 258: Screenshot of running the program

5. Tomcat started and program is running (problem solved)

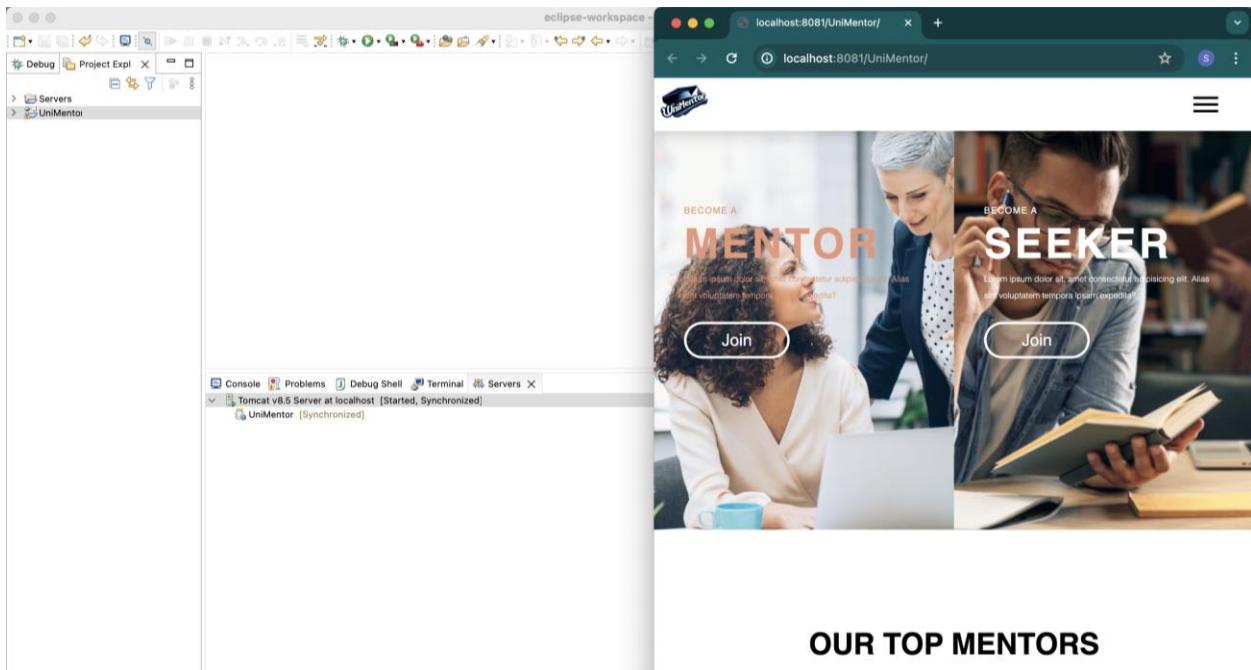


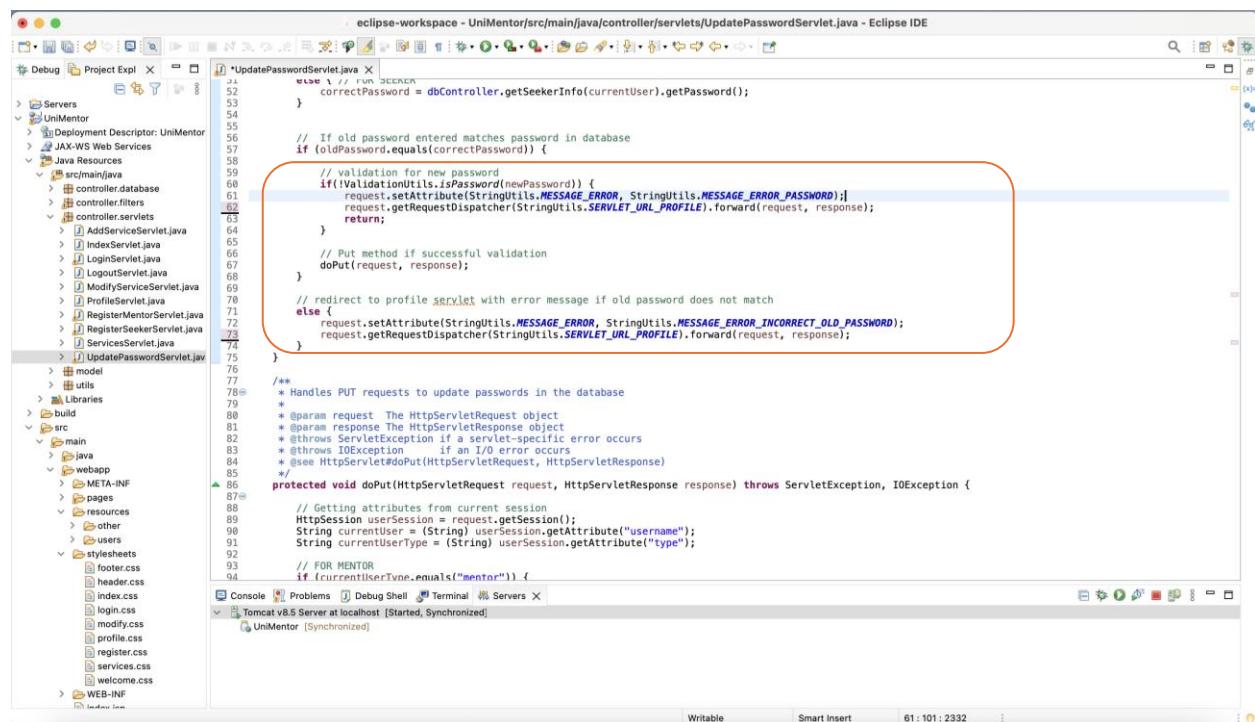
Figure 259: Screenshot of tomcat started and program running

8.2.2. Problem Faced 2 : Redirection from a servlet to another dynamic web page

Description: The profile page of my website is dynamic. When I click on update password button on my profile page, it redirects to UpdatePasswordServlet for validation and password update. When all password details are valid, the password update is successful and the servlet redirects to welcome page which is static. But in case of invalid password details for password update, the servlet should redirect to profile servlet (as profile page is dynamic). I tried using request dispatcher and send redirect but there was a problem.

Screenshots of Problem:

1. Code to redirect to profile servlet from update password servlet



```

  * @param request The HttpServletRequest object
  * @param response The HttpServletResponse object
  * @throws ServletException if a servlet-specific error occurs
  * @throws IOException if an I/O error occurs
  */
protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Getting attributes from current session
    HttpSession userSession = request.getSession();
    String currentUser = (String) userSession.getAttribute("username");
    String currentUserType = (String) userSession.getAttribute("type");
    // FOR MENTOR
    if (currentUserType.equals("mentor")) {
        // Handles PUT requests to update passwords in the database
        if (!oldPassword.equals(correctPassword)) {
            // validation for new password
            if (!ValidationUtils.isPassword(newPassword)) {
                request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_PASSWORD);
                request.getRequestDispatcher(StringUtils.SERVLET_URL_PROFILE).forward(request, response);
                return;
            }
            // Put method if successful validation
            doPut(request, response);
        }
        // redirect to profile servlet with error message if old password does not match
        else {
            request.setAttribute(StringUtils.MESSAGE_ERROR, StringUtils.MESSAGE_ERROR_INCORRECT_OLD_PASSWORD);
            request.getRequestDispatcher(StringUtils.SERVLET_URL_PROFILE).forward(request, response);
        }
    }
}

```

Figure 260: Screenshot of redirection code

2. Entering invalid input in reset password fields

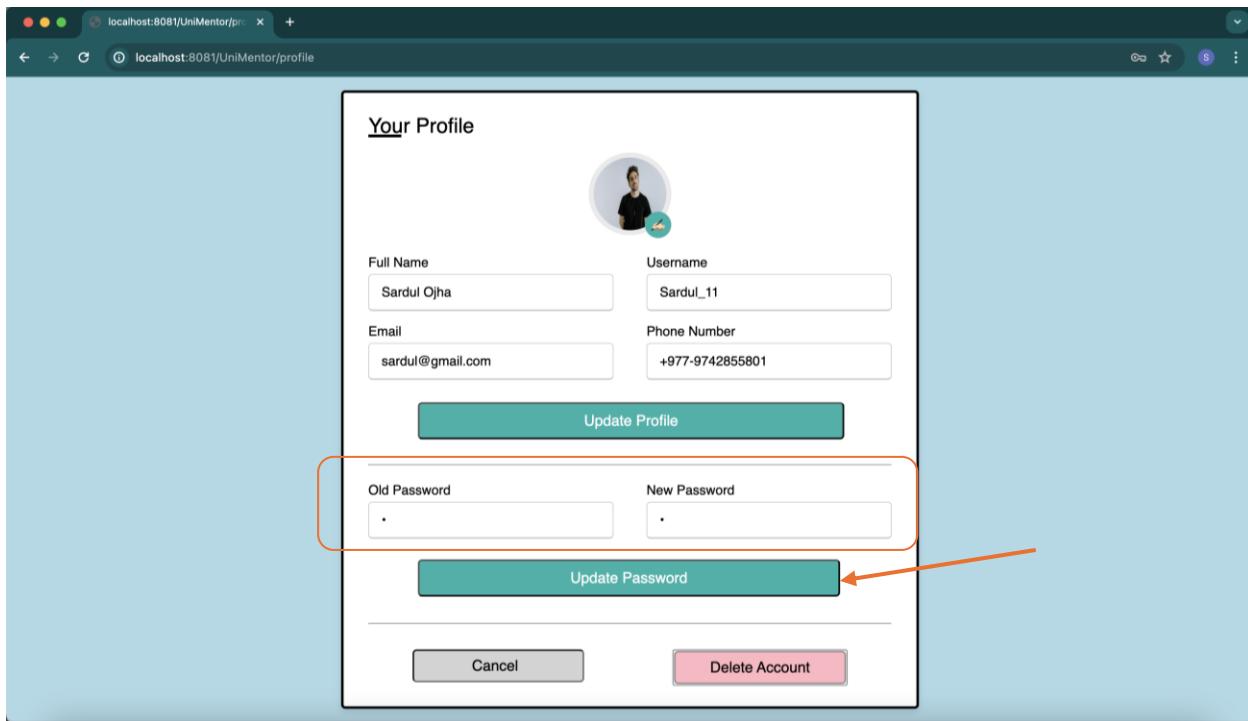


Figure 261: Screenshot of entering invalid in reset password

3. Blank page in case of exception in update password



Figure 262: Screenshot of blank page shown (error)

Solution: When I consulted my teacher and addressed this problem, he advised me to use include method to handle content from UpdatePasswordServlet to ProfileServlet. He also advised me to set the attribute “isUpdate” to 1 and handle that accordingly in the ProfileServlet. Upon changing the code, error message was displayed and program was run accordingly.

Screenshots of attempt and solution:

1. Changed code in update password servlet (include instead of forward)

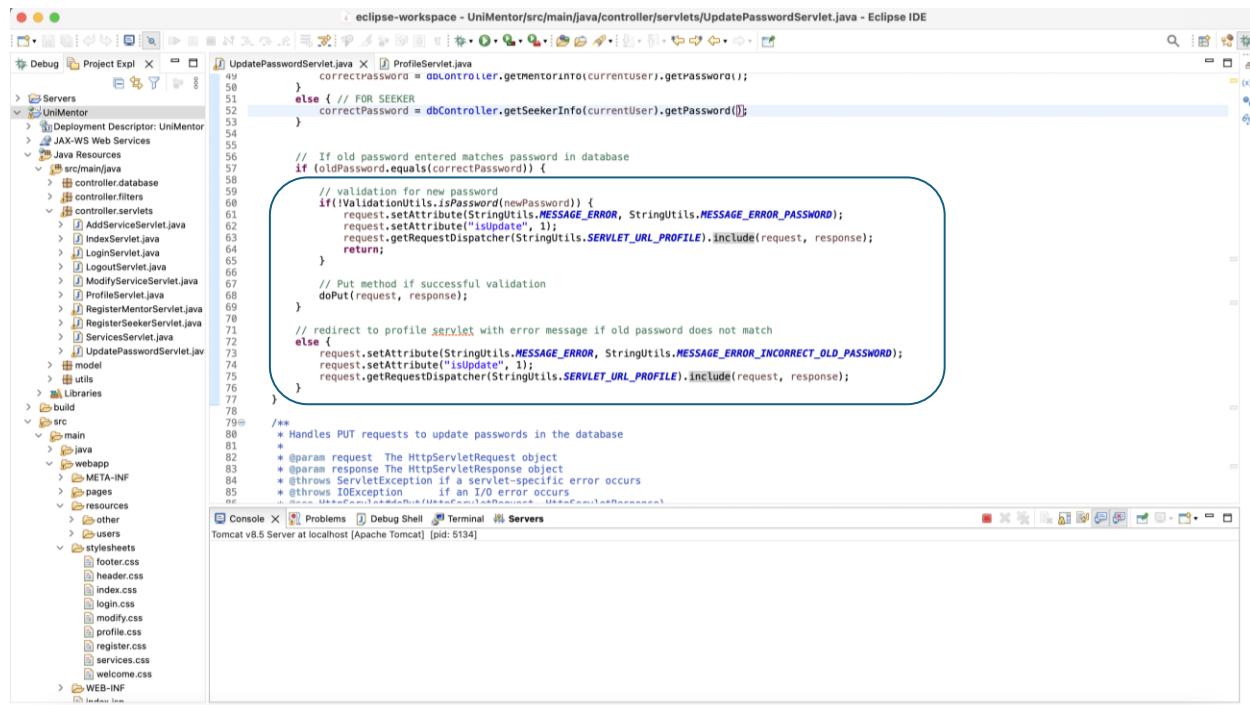


Figure 263: Screenshot of changed code

2. Changed code in profile servlet (handling isUpdate)

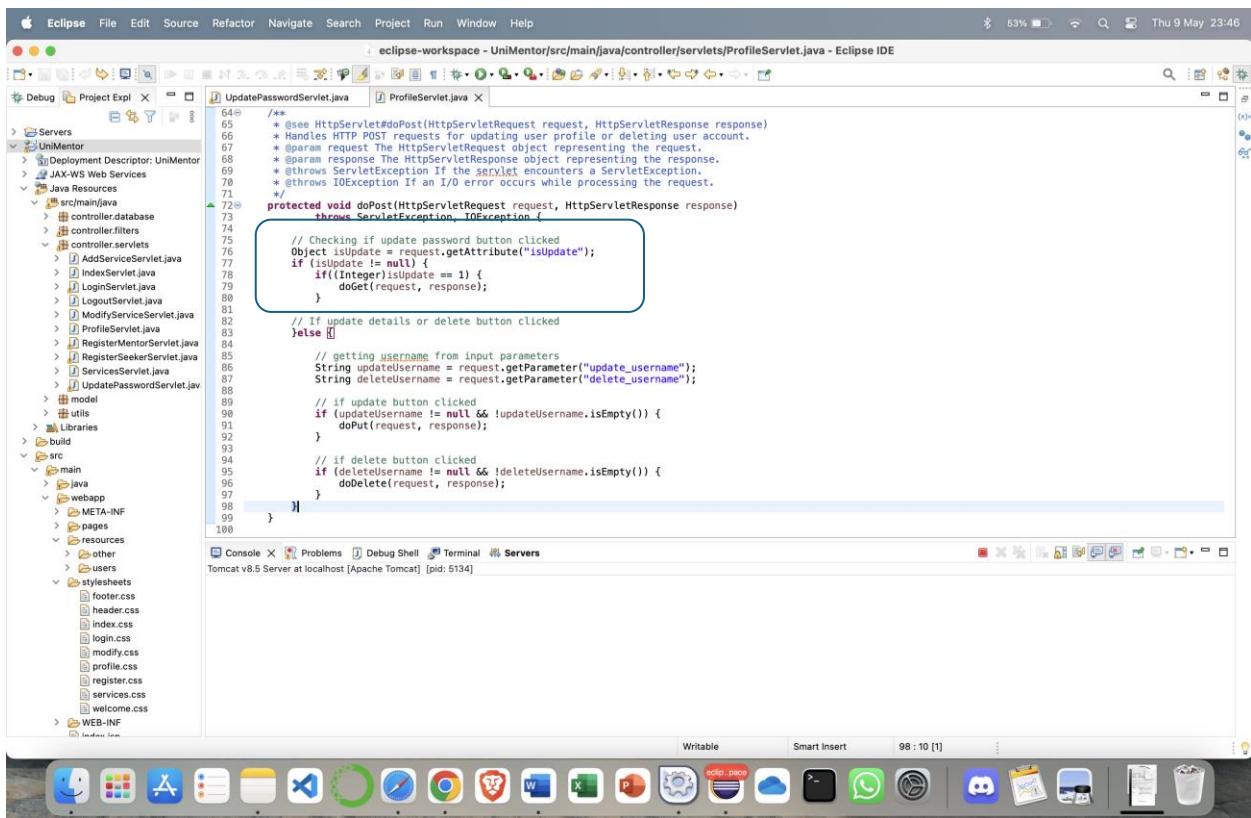


Figure 264: Screenshot of changed code in profile servlet

3. Again entering invalid input in update password

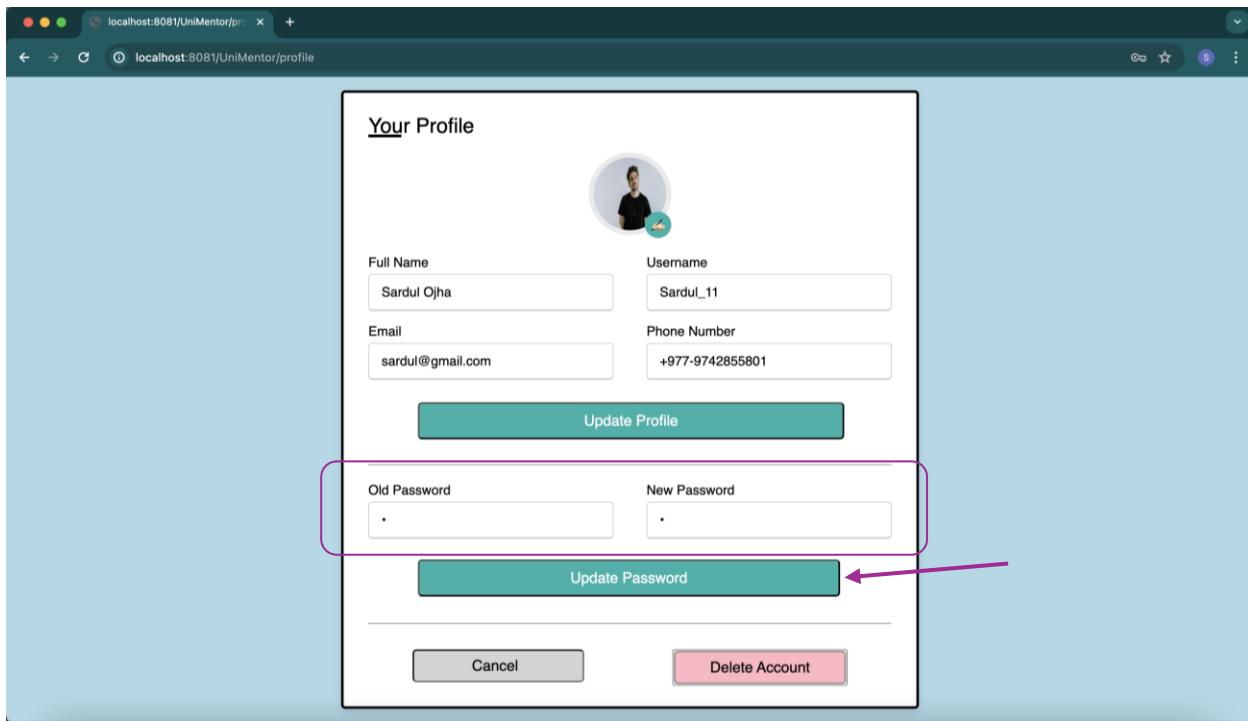


Figure 265: Screenshot of entering invalid input again in reset password

4. Error message displayed in profile page (problem solved)

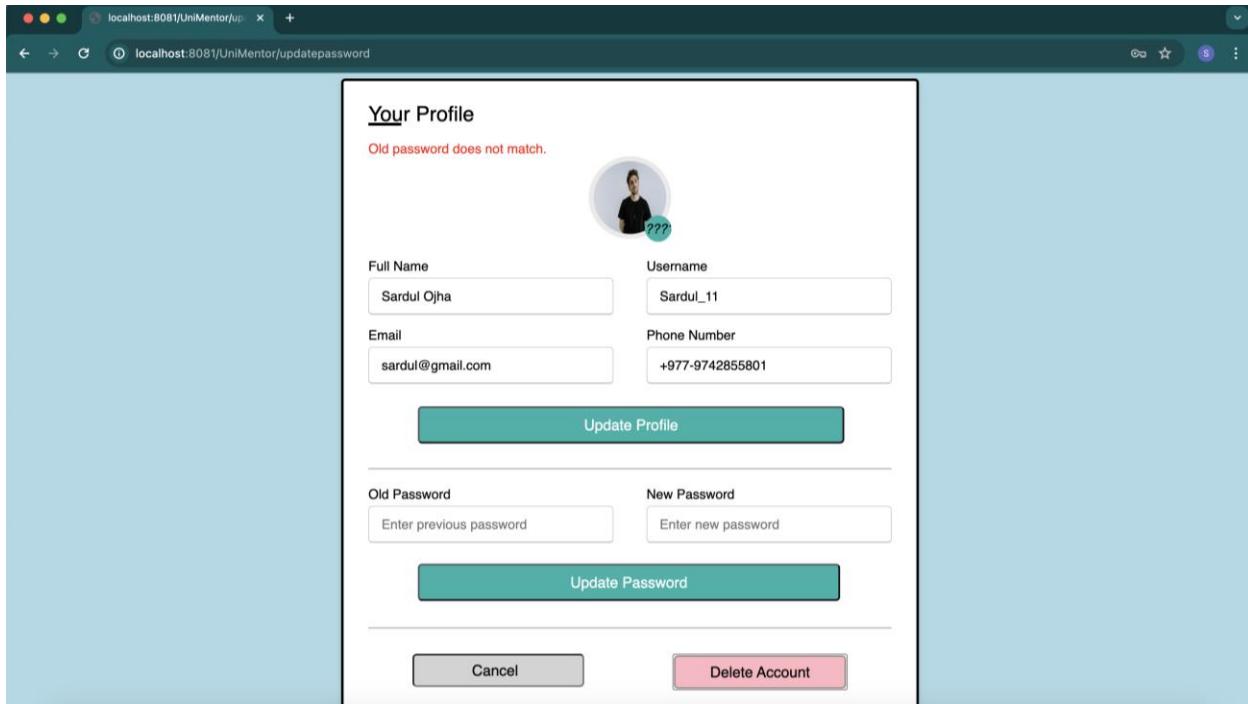


Figure 266: Screenshot of error message displayed (code working)

8.2.3. Problem Faced 3 : Error when trying to add images

Description: After completing registration process, I wanted to move on to next step i.e. adding images for registration. I did everything that was taught in week 9 for image handling. But after that when I tried registering, the following problem was faced.

Screenshots of Problem:

1. Entering valid details to register seeker

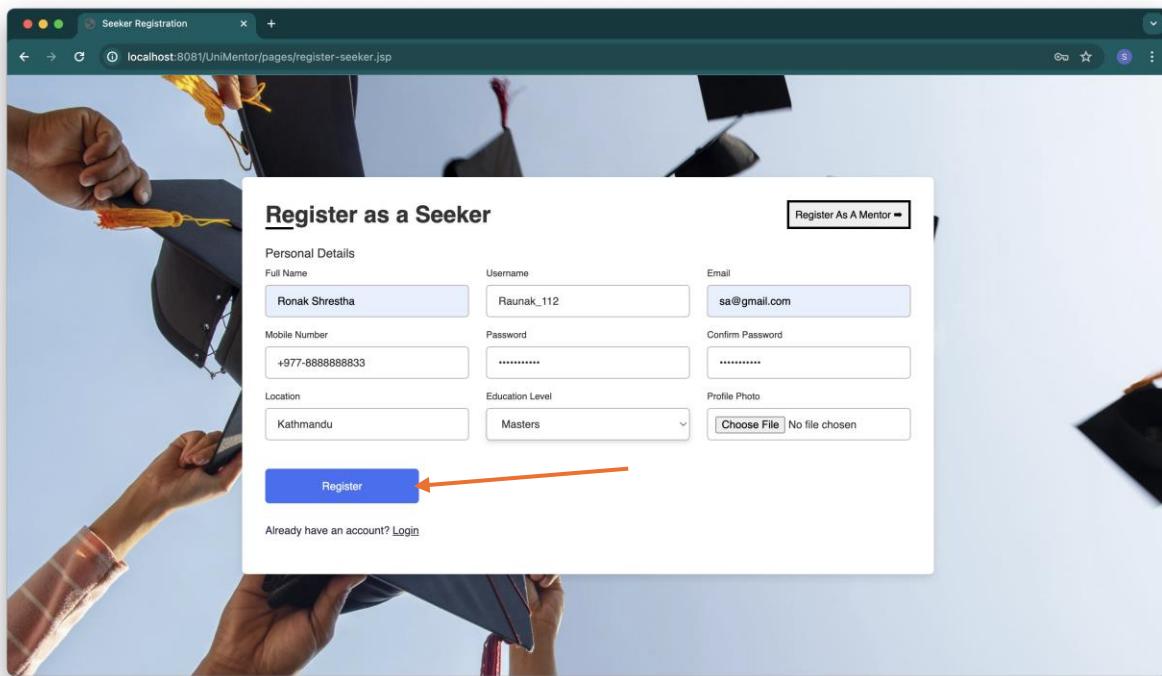


Figure 267: Screenshot of entering valid registration details

2. Error page displayed (NullPointerException)

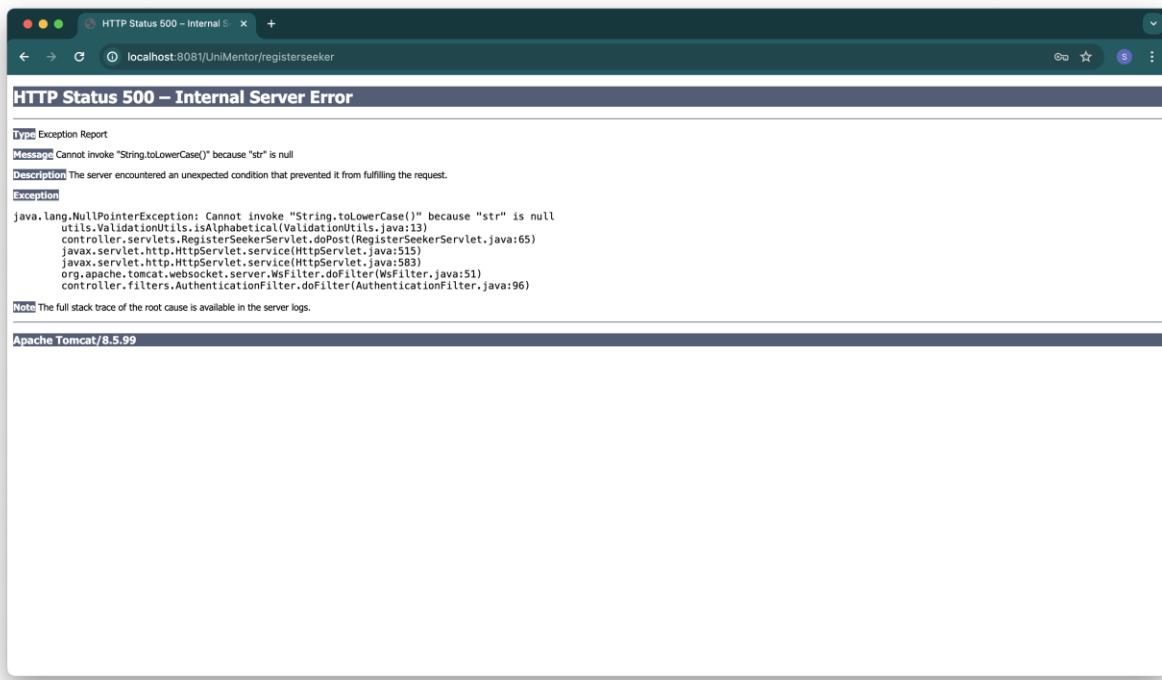


Figure 268: Screenshot of error page

Solution: On reading the error message, there was a problem in my RegisterSeekerServlet. It was also indicating null pointer exception. To tackle this problem, I ran a debugger in my register seeker servlet. On inspecting variables from my debugger it was found that all the extracted text from the input parameters from the jsp were null. So, I again started googling this problem. I ran into stack overflow([link here](#)) again to find out some annotation was missing in my servlet. I checked our teacher's code on github ([github link](#)) again to find out that "@MultiPartConfig" was missing from my servlet code. On adding the code, inserted values in the textfield were returned and registration was successful with image.

Screenshots of attempt and solution:

1. Adding breakpoints for debugging

```

eclipse-workspace - UniMentor/src/main/java/controller/services/RegisterSeekerServlet.java - Eclipse IDE

RegisterSeekerServlet.java x register-seeker.jsp

37     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
38     */
39     @Override
40     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
41         // Extract mentor information from register page request parameters
42         String name = request.getParameter(StringUtil.SEEKER_NAME);
43         String username = request.getParameter(StringUtil.SEEKER_USERNAME);
44         String email = request.getParameter(StringUtil.SEEKER_EMAIL);
45         String phone = request.getParameter(StringUtil.SEEKER_PHONE_NUMBER);
46         String password = request.getParameter(StringUtil.SEEKER_PASSWORD);
47         String retypePassword = request.getParameter(StringUtil.SEEKER_RETYPE_PASSWORD);
48         String location = request.getParameter(StringUtil.SEEKER_LOCATION);
49         String educationLevel = request.getParameter(StringUtil.SEEKER_EDUCATION_LEVEL);
50         String profilePhoto = request.getPart(StringUtil.SEEKER_PROFILE_PHOTO);
51
52         // validation flag
53         boolean isValid = true;
54
55         // message to store all error messages
56         StringBuilder errorMessage = new StringBuilder();
57
58         // full name validation
59         if (!ValidationUtils.isAlphabetical(name)) {
60             isValid = false;
61             errorMessage.append(StringUtil.MESSAGE_ERROR_NAMES);
62         }
63
64         // password validation
65         if (!ValidationUtils.isAlphabetical(password)) {
66             isValid = false;
67             errorMessage.append(StringUtil.MESSAGE_ERROR_PASSWORD);
68         }

```

Console x Problems x Debug Shell x Terminal x Servers

Tomcat v8.5 Server at localhost [Apache Tomcat/8.5.52] /Users/sardulojha/p2pool/plugins/org.eclipse.jst/openjk.hotspot.jre.full.macosx.x86_64_17.0.10.v20240120-1143/re/bin/java [May 10, 2024, 12:03:12 AM] [pid: 6308]

SEVERE: Servlet.service() for servlet [controller.services.RegisterSeekerServlet] in context with path [/UniMentor] threw exception

java.lang.NullPointerException: Cannot invoke "String.substring(int, int)" because "str" is null

at util.ValidationUtils.isAlphabetical(ValidationUtils.java:13)

at controller.services.RegisterSeekerServlet.doPost(RegisterSeekerServlet.java:65)

at javax.servlet.http.HttpServlet.service(HttpServlet.java:515)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:212)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:181)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at controller.filters.AuthenticationFilter.doFilter(AuthenticationFilter.java:99)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:181)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:98)

at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:98)

at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:483)

at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:138)

Figure 269: Screenshot of adding breakpoints for debugging

2. Debugging on server

File x New x Open x Save x Close x Project Explorer x Problems x Debug Shell x Terminal x Servers

RegisterSeekerServlet.java x register-seeker.jsp

37 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
38 */
39 @Override
40 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
41 // Extract mentor information from register page request parameters
42 String name = request.getParameter(StringUtil.SEEKER_NAME);
43 String username = request.getParameter(StringUtil.SEEKER_USERNAME);
44 String email = request.getParameter(StringUtil.SEEKER_EMAIL);
45 String phone = request.getParameter(StringUtil.SEEKER_PHONE_NUMBER);
46 String password = request.getParameter(StringUtil.SEEKER_PASSWORD);
47 String retypePassword = request.getParameter(StringUtil.SEEKER_RETYPE_PASSWORD);
48 String location = request.getParameter(StringUtil.SEEKER_LOCATION);
49 String educationLevel = request.getParameter(StringUtil.SEEKER_EDUCATION_LEVEL);
50 String profilePhoto = request.getPart(StringUtil.SEEKER_PROFILE_PHOTO);
51
52 // validation flag
53 boolean isValid = true;
54
55 // message to store all error messages
56 StringBuilder errorMessage = new StringBuilder();
57
58 // full name validation
59 if (!ValidationUtils.isAlphabetical(name)) {
60 isValid = false;
61 errorMessage.append(StringUtil.MESSAGE_ERROR_NAMES);
62 }
63
64 // password validation
65 if (!ValidationUtils.isAlphabetical(password)) {
66 isValid = false;
67 errorMessage.append(StringUtil.MESSAGE_ERROR_PASSWORD);
68 }

File x New x Open x Save x Close x Project Explorer x Problems x Debug Shell x Terminal x Servers

localhost [Apache Tomcat/8.5.52] /Users/sardulojha/p2pool/plugins/org.eclipse.jst/openjk.hotspot.jre.full.macosx.x86_64_17.0.10.v20240120-1143/re/bin/java [May 10, 2024, 12:03:12 AM] [pid: 6308]

SEVERE: Servlet.service() for servlet [controller.services.RegisterSeekerServlet] in context with path [/UniMentor] threw exception

java.lang.NullPointerException: Cannot invoke "String.substring(int, int)" because "str" is null

at util.ValidationUtils.isAlphabetical(ValidationUtils.java:13)

at controller.services.RegisterSeekerServlet.doPost(RegisterSeekerServlet.java:65)

at javax.servlet.http.HttpServlet.service(HttpServlet.java:515)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:212)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:181)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at controller.filters.AuthenticationFilter.doFilter(AuthenticationFilter.java:99)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:181)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)

at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:98)

at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:98)

at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:483)

at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:138)

Figure 270: Screenshot of running debugger

3. Registering again to check

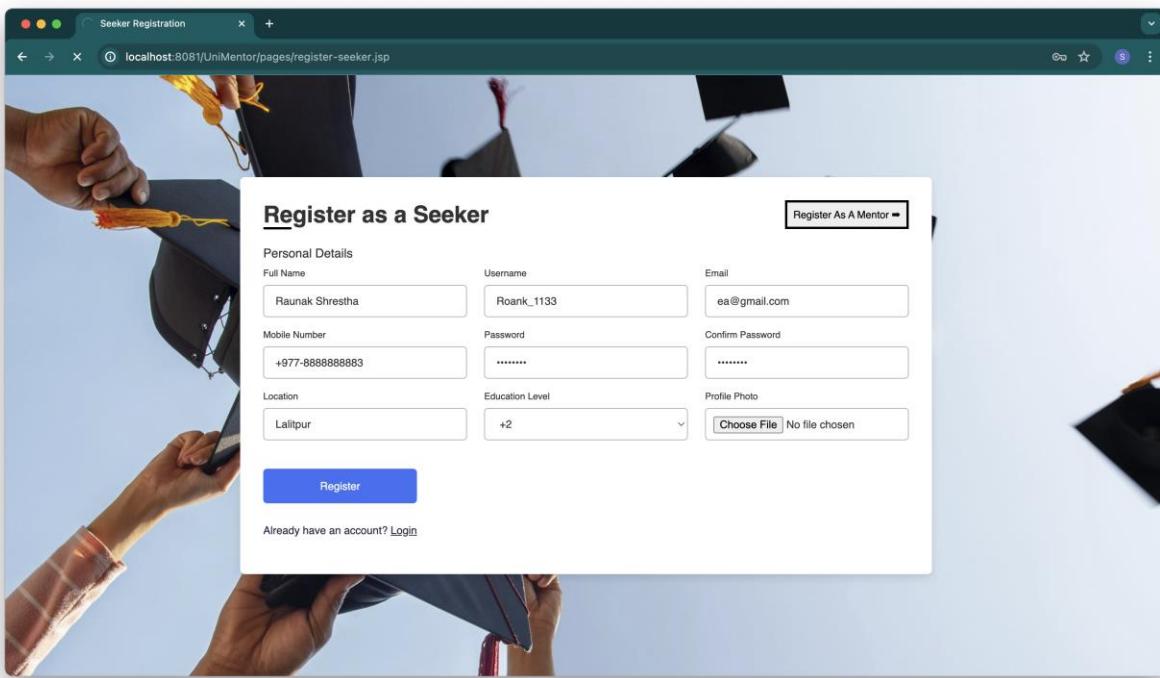


Figure 271: Screenshot of registering again

4. Program stopped on breakpoint

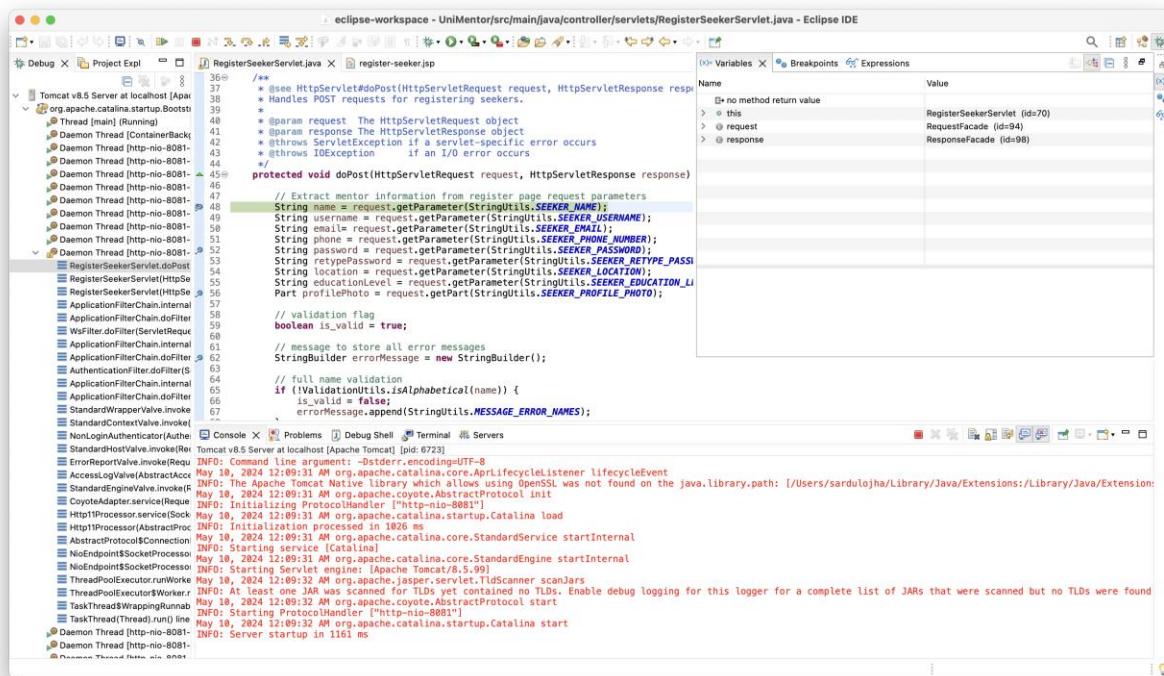


Figure 272: Screenshot of program stopped at breakpoint

5. All input parameters displaying null values

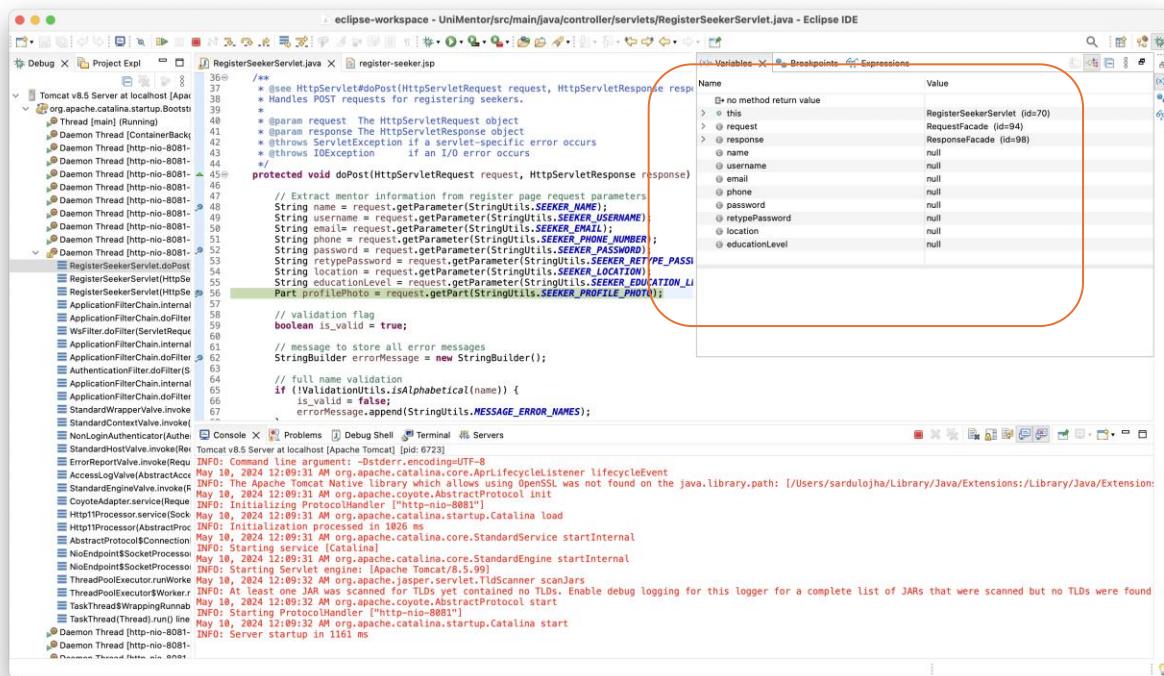


Figure 273: Screenshot of null values returned

6. Serching for solution in stack overflow

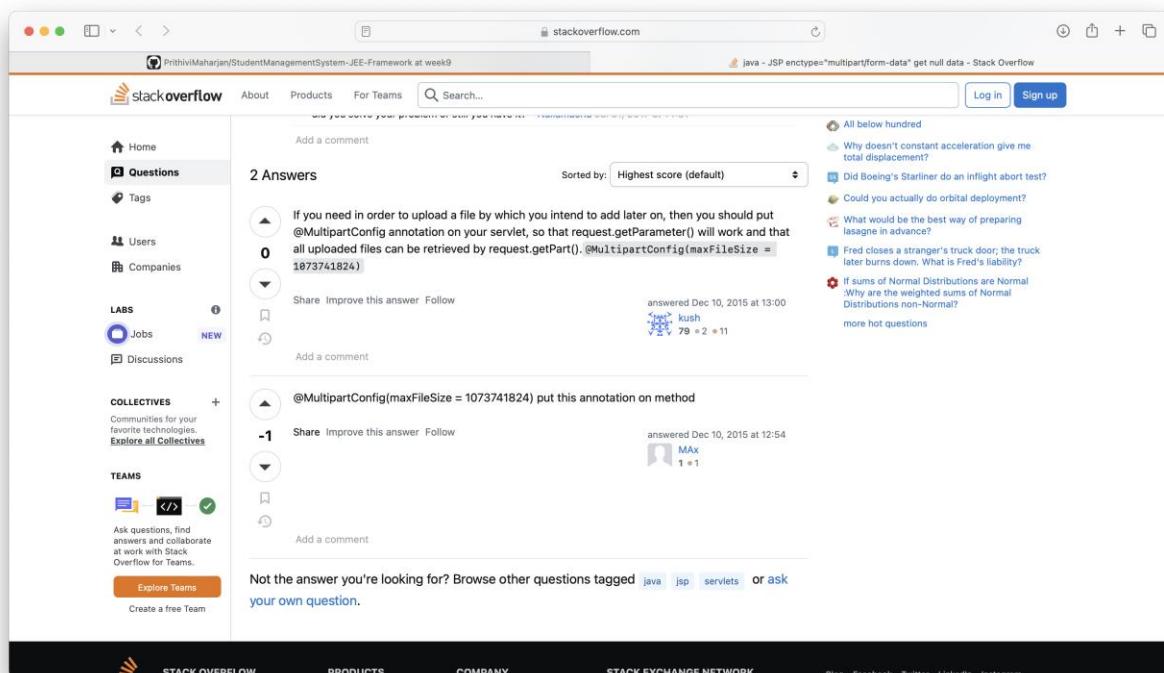


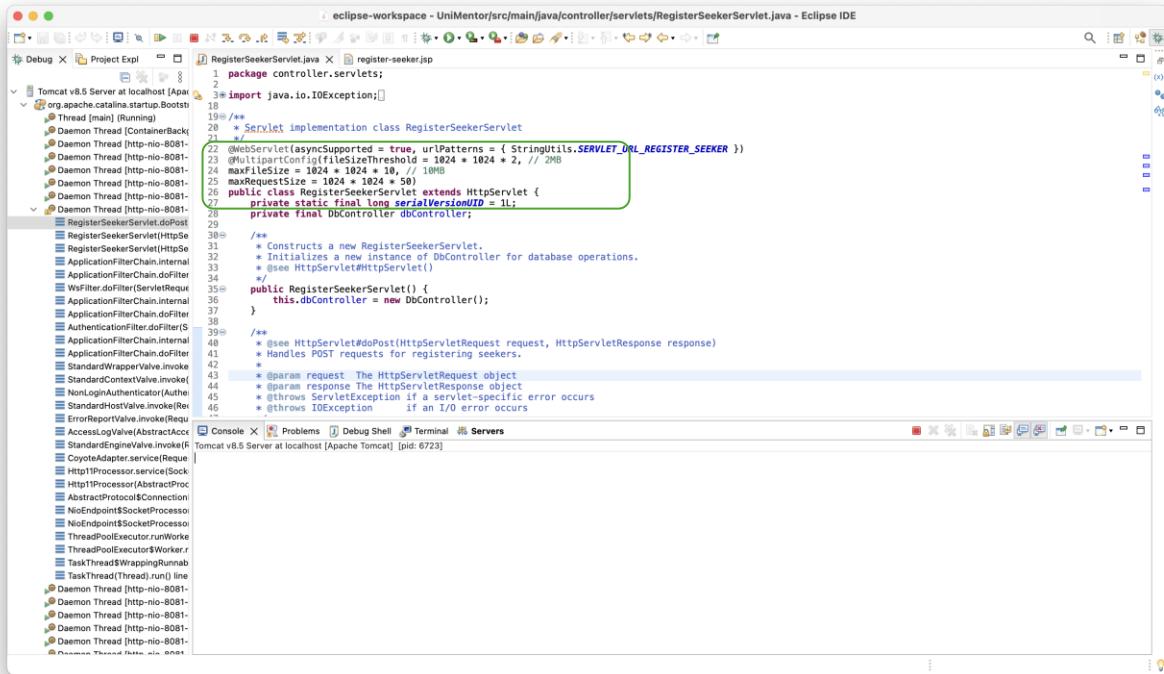
Figure 274: Screenshot of hint in stack overflow

7. Missing multipartconfig code on github

```
21      * This Servlet class handles student registration requests. It extracts student
22      * information from the registration form submission, performs basic data
23      * validation (to be implemented), and attempts to register the student in the
24      * database using a 'DBController'. The user is redirected to the login page
25      * upon successful registration.
26      *
27      * @author Prithivi Maharjan (prithivi.maharjan18@gmail.com)
28      */
29  @WebServlet(asyncSupported = true, urlPatterns = { StringUtils.SERVLET_URL_REGISTER })
30  @MultipartConfig(fileSizeThreshold = 1024 * 1024 * 2, // 2MB
31  maxFileSize = 1024 * 1024 * 10, // 10MB
32  maxRequestSize = 1024 * 1024 * 50)
33  public class RegisterStudentServlet extends HttpServlet {
34
35      private static final long serialVersionUID = 1L;
36      private final DBController dbController;
37
38      public RegisterStudentServlet() {
39          this.dbController = new DBController();
40      }
41
42      /**
43      * Handles HTTP POST requests for student registration.
44      *
45      * @param request The HttpServletRequest object containing registration form
46      *                 data.
47      * @param response The HttpServletResponse object for sending responses.
48      * @throws ServletException if a servlet-specific error occurs.
49      * @throws IOException      if an I/O error occurs.
50      */
51      @Override
52      protected void doPost(HttpServletRequest request, HttpServletResponse response)
53          throws ServletException, IOException {
54      }
```

Figure 275: Screenshot of corrected/missing code in github

8. Adding the missing code in register servlet



```

  1 package controller.servlets;
  2
  3 import java.io.IOException;
  4
  5 /**
  6  * Servlet implementation class RegisterSeekerServlet
  7  */
  8 @WebServlet(asyncSupported = true, urlPatterns = { StringTools.SERVLET_URL_REGISTER_SEEKER })
  9 public class RegisterSeekerServlet extends HttpServlet {
 10     private static final long serialVersionUID = 1L;
 11     private final DbController dbController;
 12
 13     RegisterSeekerServlet() {
 14         /*
 15          * Constructs a new RegisterSeekerServlet.
 16          * Initializes a new instance of DbController for database operations.
 17          * @see HttpServlet#HttpServlet()
 18         */
 19         dbController = new DbController();
 20     }
 21
 22     /**
 23      * Handles POST requests for registering seekers.
 24      * @param request The HttpServletRequest object
 25      * @param response The HttpServletResponse object
 26      * @throws ServletException if a servlet-specific error occurs
 27      * @throws IOException if an I/O error occurs
 28     */
 29     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 30         /*
 31          * Handles POST requests for registering seekers.
 32          * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 33         */
 34         String email = request.getParameter("email");
 35         String password = request.getParameter("password");
 36         String confirmPassword = request.getParameter("confirmPassword");
 37         String mobileNumber = request.getParameter("mobileNumber");
 38         String location = request.getParameter("location");
 39         String educationLevel = request.getParameter("educationLevel");
 40
 41         if (!password.equals(confirmPassword)) {
 42             response.setStatus(400);
 43             response.getWriter().write("Passwords do not match");
 44             return;
 45         }
 46
 47         User user = new User();
 48         user.setEmail(email);
 49         user.setPassword(password);
 50         user.setMobileNumber(mobileNumber);
 51         user.setLocation(location);
 52         user.setEducationLevel(educationLevel);
 53
 54         dbController.insertUser(user);
 55
 56         response.sendRedirect("register-seeker.jsp");
 57     }
 58
 59     /**
 60      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 61     */
 62     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 63         response.getWriter().write("GET method not supported");
 64     }
 65
 66     /**
 67      * @see HttpServlet#destroy()
 68     */
 69     protected void destroy() {
 70         dbController.close();
 71     }
 72 }

```

Figure 276: Screenshot of missing code added in servlet

9. Entering register details again for checking

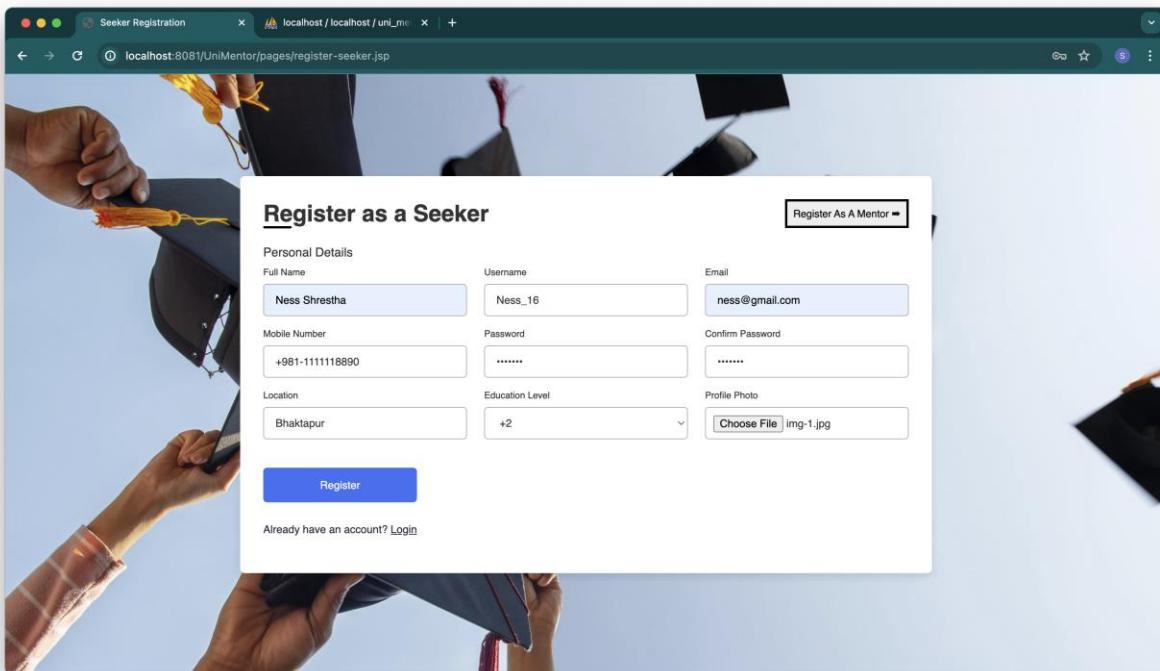


Figure 277: Screenshot of registering again

10. Register successful (Problem solved)

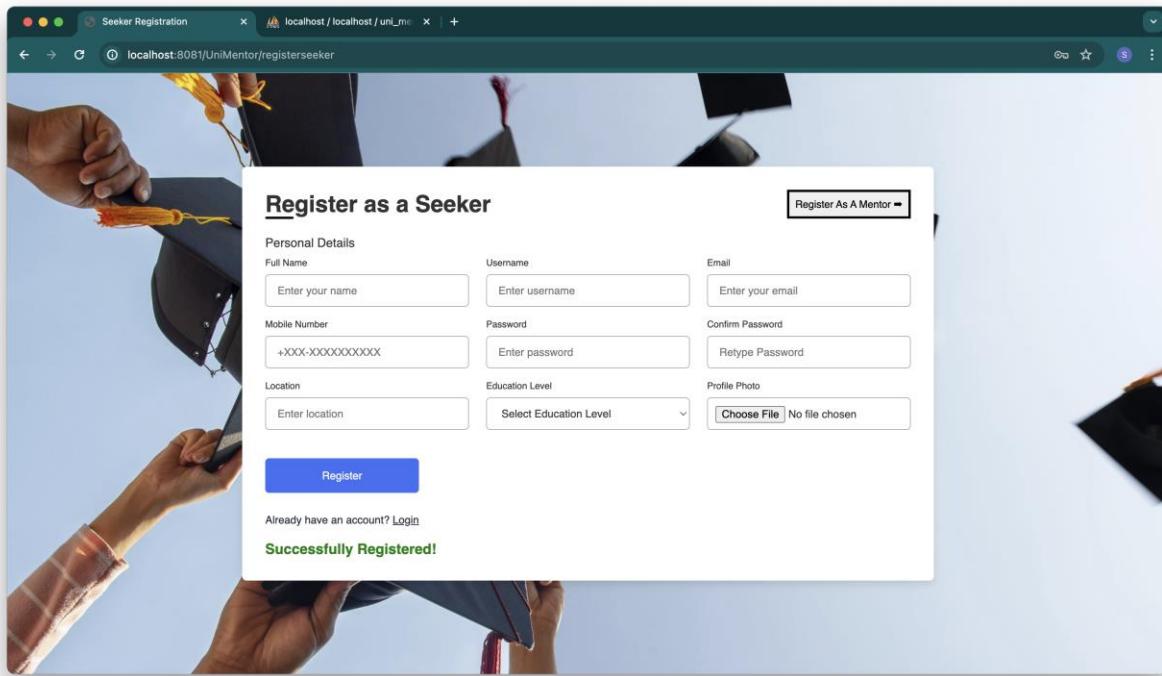


Figure 278: Screenshot of register success (problem solved)

8.3. Reflection

The development of "UniMentor" was a valuable learning experience. It provided insights into various aspects of web application development, including frontend and backend technologies, database management, and user interface design. Throughout the project, I gained a deeper understanding of Java EE server-side technology and its application in building dynamic web applications. One of the most significant lessons learned from this project was the importance of effective time management and prioritization. Managing all the work and a short deadline required too much effort, hard work, and sleepless nights to ensure the timely completion of coursework. Additionally, it also prepared me for the upcoming final year project (FYP), which is expected to be of much higher difficulty. Looking ahead, this project has given me practical skills, critical thinking abilities, and newfound confidence in my ability to tackle complex challenges. Reflecting on the strengths and weaknesses of "UniMentor", I am eager to apply the lessons learned to future projects and further refine my skills. Overall, the experience gained from developing this system has been invaluable in my journey.

9. Conclusion

A thorough evaluation of the project has been conducted in this report, and all the findings have been found to be satisfactory. All the tests that were carried out were successful, and all the project's objectives outlined in the coursework were met, with all the errors and bugs found being fixed.

This coursework has been a valuable learning experience, helping me gain knowledge, skills, and experience. I became much more familiar with Java EE and other tools such as Eclipse, Draw.io, MS Word, and many more. This real-world scenario problem also deepened my understanding of how a dynamic web application is made. Additionally, this project enhanced my research and report writing skills. All these skills and knowledge will be useful in my future endeavors.

Many challenges were encountered and solved during my coursework journey. I had to seek help from my teachers and friends to solve the problems faced. Similarly, I also faced difficulties during the report, such as formatting it correctly, crashing of MS Word and Eclipse IDE. I had to rely on Google for solutions and learned the importance of making backups after completing a major portion of the program and report.

Finally, this project has been a challenging but rewarding experience. I am proud that I was able to complete this project, and I would like to thank everyone, especially all my teachers, who helped me make this project a success. It was a great learning experience, and I am grateful to have been a part of it.

10. References

- Eclipse Foundation AISBL, n.d. *Eclipse Foundation*. [Online]
Available at: <https://www.eclipse.org/home/whatis/>
- LogicMonitor Inc, 2024. *blog*. [Online]
Available at: <https://www.logicmonitor.com/blog/what-is-apache-tomcat-server-and-how-does-it-work>
- Apache Friends, 2023. *XAMPP installers*. [Online]
Available at: <https://www.apachefriends.org>
- BYJU'S, 2021. *Byjus*. [Online]
Available at: <https://byjus.com/govt-exams/microsoft-word/>
- Hope, Computer, 2020. *ComputerHope*. [Online]
Available at: computerhope.com/jargon/d/drawio.htm
- Wikimedia Foundation, Inc, 2023. *Wikipedia*. [Online]
Available at: [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
- The University of Texas at Austin, 2021. *Fundamental Front End Languages*. [Online]
Available at: <https://techbootcamps.utexas.edu/blog/html-css-javascript/#:~:text=Like%20we%20mentioned%20earlier%2C%20HTML,a%20user%20can%20interact%20with.>
- Oracle, 2024. *documentation*. [Online]
Available at: <https://dev.mysql.com/doc/refman/8.3/en/what-is-mysql.html>
- IBM, 2023. *JSP and Java servlet programming*. [Online]
Available at: <https://www.ibm.com/docs/en/i/7.3?topic=java-jsp-servlet-programming>
- Progress Software Corporation, 2024. *FAQs*. [Online]
Available at: <https://www.progress.com/faqs/datadirect-jdbc-faqs/what-is-a-jdbc-driver#:~:text=A%20JDBC%20driver%20uses%20the,with%20a%20Java%20Virtual%20Machine.>