



University of
KURDISTAN
Hewlêr

**Student Talent Development Center:
Design and Implementation of Scheduling and
Resources Management Web Application System**

By:

Muhammad Nawzad Abdullah

University of Kurdistan Hewlêr
Erbil, Kurdistan

BSc

June, 2023



University of
KURDISTAN
Hewlêr

**Student Talent Development Center:
Design and Implementation of Scheduling and
Resources Management Web Application System**

By
Muhammad Nawzad Abdullah

Student Number: 01-19-00152

A thesis submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Software Engineering

Department of Computer Science and Engineering
School of Science and Engineering
University of Kurdistan Hewlêr

BSc
June, 2023
Erbil, Kurdistan

Declaration

I hereby declare that this dissertation/thesis entitled: "Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System" is my own original work and hereby certify that unless stated, all work contained within this is my own independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgment is made in the text.

Signature:

Name: Muhammad Nawzad Abdullah

Date:

Supervisor's Certificate

This dissertation/thesis has been written under my supervision and has been submitted for the award of the degree of 'Bachelor of Science' in 'Software Engineering' with my approval as supervisor.

Signature

Name

Date

I confirm that all the requirements have been fulfilled.

Signature

Name

Head of Department of

Date

I confirm that all the requirements have been fulfilled.

Signature

Name

Dean of School of

Date

Examining Committee Certification

We certify that we have read this dissertation/thesis: "Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System" and as a committee have examined the student: "Muhammad Nawzad Abdullah" in its content and what is related to it. We approve that it meets the standards of a dissertation/thesis for the degree of BSc in Software Engineering.

Signature

Name:

Chairman

Date

Signature

Name:

Supervisor

Date

Signature

Name:

Member

Date

Dedication

This paper is dedicated to my dear family and loved ones, who have supported me throughout my undergraduate degree.

I'd also like to dedicate this paper to Dr. Hossein Hassani in particular for all of his guidance and assistance. It is thanks to his guidance that I have learned so much and progressed so far as a student and as a person.

I'd also like to dedicate this paper to all of my colleagues and teachers who have supported me over the past 4 years.

Muhammad Nawzad Abdullah

2023

Acknowledgments

In particular, I'd like to take this opportunity to thank Dr. Hossein, my supervisor, for all of his hard work and dedication this past year. Your insightful ideas and advice were invaluable to me as I completed the project. For this, you have my undying gratitude.

I also want to express my gratitude to all of my friends and classmates for all the insightful conversations we have had, during which we shared our thoughts and ideas and shared a lot of memories together in which helped me grow as a person.

In addition, I'd like to express my gratitude to everyone at work (DIT), and among my friends who have taught me anything directly or indirectly that has helped me out with this project.

I'd also like to express my gratitude to every professor I've had over the last four years.

To conclude, I want to thank my family and loved ones who have always supported me to learn.

Thank You.

Abstract

As the importance of technology in teaching and learning in the modern university grows, more and more specialized software is being developed for use in this field. One example is the rise in popularity of scheduling and resource management systems among educational institutions. Automating tasks that would otherwise need a lot of time and manual input, such systems help accelerate the scheduling process for both the staff and students. The goal of this project is to automate the previously manual processes at STDC by creating a web-based software with a proper database. Students will be able to verify session availability and book sessions with the help of the system. Furthermore, the system will give students access to any files or lectures that have been submitted by their tutors. The system's scheduling method will be beneficial for both students and their designated tutors.

Table of Contents

Declaration.....	II
Supervisors Certificate.....	III
Examining Committee Certification	IV
Dedication	V
Acknowledgments	VI
Abstract	VII
Table of Contents	VIII
List of Tables	IX
List of Figures.....	XX
List of Abbreviations	XXI
Chapter 1 Introduction	1
1.1 Problem Statement.....	2
1.2 Objectives	3
1.3 Thesis Organisation	3
Chapter 2 Background and Literature Review	4
Chapter 3 Methodology	9
3.1 Software and Hardware Requirements.....	14
Chapter 4 Analysis and Design	16
4.1 Planning	16
4.1.1 Stakeholder Identification.....	16

4.1.2 Activity Identification	16
4.1.3 Task Duration Identification.....	17
4.1.4 Work Breakdown Structure	18
4.1.5 Identify Critical Path	19
4.1.6 Risk Analysis	22
4.2 Analysis.....	27
4.2.1 Requirement Elicitation (Requirements Gathering)	27
4.2.2 Requirement Specification	34
4.2.3 Requirement Validation.....	43
4.3 Design	44
4.3.1 Architecture Design.....	44
4.3.2 Program Design	50
4.3.3 Interface Design	56
4.3.4 Database Design	68
Chapter 5 Implementation and Testing	74
5.1 Implementation	75
5.1.1 Backend Implementation (API).....	77
5.1.2 Frontend Implementation (Client)	126
5.2 Testing	211
5.2.1 Test Plan.....	211
5.2.2 Requirements Traceability Matrix.....	213
5.2.3 Requirement Specifications Testing	221

5.2.4 Model Based Testing	223
5.2.5 Boundary Analysis Testing	227
5.2.6 Inspection Testing	231
5.2.7 Granularity of Automated Testing.....	238
5.2.8 Unit Testing	240
5.2.9 Integration Testing.....	259
5.2.10 Build Verification Testing	268
5.2.11 Usability Testing	272
5.2.12 User Acceptance Testing.....	292
5.3 Deployment.....	297
5.3.1 Backend Deployment (API).....	297
5.3.2 Frontend Deployment (Client)	302
Chapter 6 Conclusion	304
6.1 Summary	304
6.2 Future Works	305
References.....	R1
Appendices	A1

List of Tables

Table 3.1 MoSCoW prioritization	12
Table 4.1 Activities duration	18
Table 4.2 Activity dependency table.....	19
Table 4.3 Use case tabular form – fields description	38
Table 4.4 Class design patterns	53
Table 5.1 Requirements Traceability Matrix (RTM) - 1	214
Table 5.2 Requirements Traceability Matrix (RTM) - 2	215
Table 5.3 Requirements Traceability Matrix (RTM) - 3	216
Table 5.4 Requirements Traceability Matrix (RTM) - 4	217
Table 5.5 Requirements Traceability Matrix (RTM) - 5	218
Table 5.6 Requirements Traceability Matrix (RTM) - 6	219
Table 5.7 Requirements Traceability Matrix (RTM) - 7	220
Table 5.8 FSM State Transition Table (c: current node, n: next node)	225
Table 5.9 Actions and their references	226
Table 5.10 Numeric boundary analysis - test cases table	229
Table 5.11 Requirements Testing Table	293
Table 5.12 Environment Variables.....	298
Table 5.13 Environment Variables.....	302

List of Figures

Figure 3.1 Waterfall approach	9
Figure 3.2 Difference between incremental and iterative.....	10
Figure 3.3 Implementation approach	13
Figure 4.1 Work breakdown structure	19
Figure 4.2 Network diagram	20
Figure 4.3 Critical path (pert chart)	21
Figure 4.4 Use case model - conceptual system overview.....	35
Figure 4.5 Use case model - system core.....	36
Figure 4.6 Use case model - scheduling system	36
Figure 4.7 Use case model - feedback system.....	37
Figure 4.8 Use case model - resource management system	37
Figure 4.9 Activity model - view session	40
Figure 4.10 Activity model - provide feedback	41
Figure 4.11 Use case model - upload material	42
Figure 4.12 Physical separation	44
Figure 4.13 User interaction	46
Figure 4.14 Logical layer	47
Figure 4.15 MVC pattern	48
Figure 4.16 Class model – dynamic and static classes	51
Figure 4.17 Class model – static classes.....	52

Figure 4.18 Sequence model – identity provider in action	54
Figure 4.19 Sequence model – setting the current user	55
Figure 4.20 UI01. landing page (login)	57
Figure 4.21 UI02. home page	58
Figure 4.22 UI03. profile page.....	59
Figure 4.23 UI04. modules page	60
Figure 4.24 UI05. sessions page.....	61
Figure 4.25 UI06. feedbacks page	62
Figure 4.26 UI07. venues page	63
Figure 4.27 UI08. users page	64
Figure 4.28 UI09. view module page.....	65
Figure 4.29 UI10. view session page	66
Figure 4.30 UI11. form page	67
Figure 4.31 Database – logical entity attribute relationship diagram	70
Figure 4.32 Database – physical entity attribute relationship diagram	73
Figure 5.1 OAuth2.0 Authorization Code Flow - STDCs Authentication Process	82
Figure 5.2 Authentication - GuardController.....	83
Figure 5.3 Authentication - CurrentUserConfigurable	84
Figure 5.4 Authentication - AuthorizationService.....	85
Figure 5.5 Authentication - JsonWebToken	87
Figure 5.6 Authentication - UsersService	89
Figure 5.7 Authorization - Ability	92

Figure 5.8 Authorization - m2m_abilities	93
Figure 5.9 Authorization - superadmin_abilities	94
Figure 5.10 Authorization - tutor_abilities	95
Figure 5.11 Authorization - student_abilities.....	96
Figure 5.12 Course Model - Enums	98
Figure 5.13 Course Model - Methods	99
Figure 5.14 Course Model - Migration File	100
Figure 5.15 Course Model - Associations	100
Figure 5.16 Course Model - Validations	101
Figure 5.17 Course Model - Callback	102
Figure 5.18 Course Model - Count Cachable Concern.....	102
Figure 5.19 Course Model - Scopes.....	103
Figure 5.20 ApplicationController.....	105
Figure 5.21 Crudable - Part 1	106
Figure 5.22 Crudable - Part 2	107
Figure 5.23 PaginationHelper	108
Figure 5.24 CoursesController - Part 1	110
Figure 5.25 CoursesController - Part 2	111
Figure 5.26 BaseSerializer	112
Figure 5.27 SerializersHelper	113
Figure 5.28 CourseSerializer	114
Figure 5.29 CustomResponseExceptionSerializer	115

Figure 5.30 ErrorRespondable	116
Figure 5.31 JsonErrors	116
Figure 5.32 CustomExceptions.....	117
Figure 5.33 Swagger - Part 1.....	120
Figure 5.34 Swagger - Part 2.....	121
Figure 5.35 Swagger - Part 3.....	122
Figure 5.36 Swagger - Part 4.....	123
Figure 5.37 Swagger - Course Posts Request and Response	124
Figure 5.38 Auth0 Dashboard - Application Keys	130
Figure 5.39 Auth0 Dashboard - Allowed URLs.....	131
Figure 5.40 Index.js	132
Figure 5.41 Landing Page - Login Code Block	134
Figure 5.42 NavBar - Logout Code Block	135
Figure 5.43 App.js	137
Figure 5.44 Routes - App.js	139
Figure 5.45 Public Routes - routes.js.....	141
Figure 5.46 Protected Routes - routes.js	142
Figure 5.47 Protected Routes - hasAccessTo Usage	143
Figure 5.48 hasAccessTo.....	144
Figure 5.49 hasAccessTo - Sidebar Example	145
Figure 5.50 hasAccessTo - Panel Example Part 1	146
Figure 5.51 hasAccessTo - Panel Example Part 2	147

Figure 5.52 OpenAPI Command to Generate Services	148
Figure 5.53 OpenAPI Base Address	149
Figure 5.54 Services Folder	149
Figure 5.55 Services Usage - Get Courses	150
Figure 5.56 Generated Service - Post Course Endpoint	150
Figure 5.57 React Query Configuration	151
Figure 5.58 Services Query Usage.....	152
Figure 5.59 Axios Interceptors Configuration.....	153
Figure 5.60 Store	155
Figure 5.61 Root Reducer.....	156
Figure 5.62 Course Slice	158
Figure 5.63 Venue Schema	161
Figure 5.64 Venue Columns - Part 1	164
Figure 5.65 Venue Columns - Part 2	165
Figure 5.66 Venue Page Header	166
Figure 5.67 Venue Table - Part 1	168
Figure 5.68 Venue Table - Part 2	169
Figure 5.69 Venue Table - Part 3	170
Figure 5.70 Venue Index	171
Figure 5.71 Create Venue - Part 1	173
Figure 5.72 Create Venue - Part 2	174
Figure 5.73 Create Venue - Part 3	175

Figure 5.74 Update Venue - Part 1	177
Figure 5.75 Update Venue - Part 2	178
Figure 5.76 Update Venue - Part 3	179
Figure 5.77 Login - Landing Page	181
Figure 5.78 Login - Auth0 Universal Login Page	182
Figure 5.79 Login - Microsoft Login Page	182
Figure 5.80 Login - Grant Permissions	183
Figure 5.81 Home Page	184
Figure 5.82 Home Page - Viewer Perspective	185
Figure 5.83 Users Page - Users Table	185
Figure 5.84 Users - Onboard Page.....	186
Figure 5.85 Users - Student Profile Page	186
Figure 5.86 Users - Tutor Profile Page - 1.....	187
Figure 5.87 Users - Tutor Profile Page - 2.....	187
Figure 5.88 Users - Viewer Profile Page	188
Figure 5.89 Venues Page - Venues Table	189
Figure 5.90 Venues - Create Venue Page.....	190
Figure 5.91 Venues - Create Venue Page With Validation	190
Figure 5.92 Deactivate Modal	191
Figure 5.93 Activate Modal	191
Figure 5.94 Courses Page - Courses Table	192
Figure 5.95 Courses - Create Course Page.....	192

Figure 5.96 Courses - Show Course Page	193
Figure 5.97 Courses - Show Course Page - Properties Tab	193
Figure 5.98 Courses - Assign Tutor Page	194
Figure 5.99 Feedbacks Page - Feedbacks Table	195
Figure 5.100 Feedbacks Page - View Feedback.....	195
Figure 5.101 Provide Feedback from Sessions Show Page	196
Figure 5.102 Questions Page - Questions Table	197
Figure 5.103 Questions Page - Create Question.....	197
Figure 5.104 Sessions Page - Card View	198
Figure 5.105 Sessions Page - Table View.....	198
Figure 5.106 Sessions Page - Session History.....	199
Figure 5.107 Sessions Page - Create Session Page With Validation	200
Figure 5.108 Sessions Page - Show Session Page.....	200
Figure 5.109 Sessions Page - Show Session Page - Viewer Perspective	201
Figure 5.110 Sessions Page - Show Session Page - Student Perspective	201
Figure 5.111 Show Session Page - Attendance Tab	202
Figure 5.112 Show Session Page - Registered Students Tab.....	202
Figure 5.113 Show Session Page - Assign Time and Venue Tab	203
Figure 5.114 Show Session Page - Upload Material Tab.....	203
Figure 5.115 Show Session Page - Materials Tab.....	204
Figure 5.116 Enter Availability Page.....	205
Figure 5.117 Enter Availability Page.....	205

Figure 5.118 Logout Button.....	206
Figure 5.119 Release Notes Page.....	206
Figure 5.120 Register Modal	207
Figure 5.121 404 Not Found Page	207
Figure 5.122 403 Forbidden Page.....	208
Figure 5.123 FSM State Transitions.....	224
Figure 5.124 Invalid course name in courses table	228
Figure 5.125 Invalid course name in course profile	228
Figure 5.126 Numeric boundary analysis	229
Figure 5.127 Created venues with invalid capacities.....	230
Figure 5.128 Inspection testing - HTTP 403 when login	232
Figure 5.129 Inspection testing - Case One Defected Line	233
Figure 5.130 Inspection testing - Case One Solution	234
Figure 5.131 Inspection testing - Case Two defective search.....	235
Figure 5.132 Inspection testing - Case Two defected line.....	236
Figure 5.133 Inspection testing - Case Two Solution.....	236
Figure 5.134 User Model - Validation Unit Tests	240
Figure 5.135 User Model - Association Unit Tests	241
Figure 5.136 User Model Unit Tests	242
Figure 5.137 Session Model - Validation Unit Tests	242
Figure 5.138 Session Model - Association Unit Tests	243
Figure 5.139 Session Model Unit Tests	244

Figure 5.140 Venue Model - Validation Unit Tests.....	244
Figure 5.141 Venue Model - Association Unit Tests.....	245
Figure 5.142 Venue Model Unit Tests	246
Figure 5.143 Question Model - Validation Unit Tests.....	246
Figure 5.144 Question Model - Association Unit Tests.....	247
Figure 5.145 Question Model Unit Tests	247
Figure 5.146 Feedback Model - Validation Unit Tests.....	248
Figure 5.147 Feedback Model - Association Unit Tests.....	248
Figure 5.148 Feedback Model Unit Tests	249
Figure 5.149 Material Model - Validation Unit Tests	250
Figure 5.150 Material Model - Association Unit Tests	251
Figure 5.151 Material Model Unit Tests	251
Figure 5.152 Enrollment Model - Association Unit Tests	252
Figure 5.153 Enrollment Model Unit Tests	253
Figure 5.154 Course Model - Validation Unit Tests.....	253
Figure 5.155 Course Model - Association Unit Tests.....	254
Figure 5.156 Course Model Unit Tests	255
Figure 5.157 Attendance Model - Validation Unit Tests	255
Figure 5.158 Attendance Model - Association Unit Tests	256
Figure 5.159 Attendance Model Unit Tests	256
Figure 5.160 Answer Model - Validation Unit Tests	257
Figure 5.161 Answer Model - Association Unit Tests.....	257

Figure 5.162 Answer Model Unit Tests	258
Figure 5.163 Automated Integration Testing - Setup	259
Figure 5.164 Course Integration Testing - Get All Courses	261
Figure 5.165 Course Integration Testing - Get Course By ID	262
Figure 5.166 Course Integration Testing - Create Courses	262
Figure 5.167 Course Integration Testing - Update Course	263
Figure 5.168 Course Integration Testing - Get All Courses With No Pagination .	264
Figure 5.169 Course Integration Testing - Get All Students	265
Figure 5.170 Course Integration Testing - Assign Tutor.....	266
Figure 5.171 Build Verification Testing - Failed Scenario.....	269
Figure 5.172 Build Verification Testing - Succeeded Scenario	270
Figure 5.173 Build Verification Testing - CI/CD YAML.....	271
Figure 5.174 Usability Testing Questionnaires - Guide.....	274
Figure 5.175 Usability Testing Questionnaires - Section I.....	275
Figure 5.176 Usability Testing Questionnaires - Section II (1)	275
Figure 5.177 Usability Testing Questionnaires - Section II (2)	276
Figure 5.178 Usability Testing Questionnaires - Section II (3)	276
Figure 5.179 Usability Testing Questionnaires - Section II (4)	277
Figure 5.180 Usability Testing Questionnaires - Section II (5)	277
Figure 5.181 Usability Testing Questionnaires - Section II (6)	278
Figure 5.182 Usability Testing Questionnaires - Section II (7)	278
Figure 5.183 Usability Testing Results - Question 1.....	279

Figure 5.184 Usability Testing Results - Question 2.....	279
Figure 5.185 Usability Testing Results - Question 3.....	280
Figure 5.186 Usability Testing Results - Question 4.....	280
Figure 5.187 Usability Testing Results - Question 5.....	281
Figure 5.188 Usability Testing Results - Question 6.....	281
Figure 5.189 Usability Testing Results - Question 7.....	282
Figure 5.190 Usability Testing Results - Question 8.....	282
Figure 5.191 Usability Testing Results - Question 9.....	283
Figure 5.192 Usability Testing Results - Question 10	283
Figure 5.193 Usability Testing Results - Question 11	284
Figure 5.194 Usability Testing Results - Question 12	284
Figure 5.195 Usability Testing Results - Question 13	285
Figure 5.196 Usability Testing Results - Question 14	286
Figure 5.197 Usability Testing Results - Question 15	287
Figure 5.198 Usability Testing Results - Question 16	288
Figure 5.199 Usability Testing Results - Question 17	289
Figure 5.200 Procfile	300

List of Abbreviations

UKH	University of Kurdistan Hewlêr.
STDC	Student Talent Development Center.
PAL	Peer Assisted Learning.
SDLC	Software Development Life Cycle.
LMS	Learning Management System.
LAMP	Linux, Apache, MySQL, PHP.
KRI	Kurdistan Region of Iraq
UI	User Interface.
SQL	Standard Query Language.
CSS	Cascading Style Sheets.
HTML	HyperText Markup Language.
JS	JavaScript.
PC	Personal Computer.
LATEX	Lamport's TeX.
API	Application Programming Interface.
EARD	Entity Attribute Relationship Diagram.
WBS	Work Breakdown Structure.
JWT	JSON Web Token.
UML	Unified Modeling Language.
B2C	Business to Consumer.
AWS	Amazon Web Services.
MVC	Model View Controller.

JSON	JavaScript Object Notation.
OOP	Object Oriented Programming.
DBMS	Database Management System.
RDBMS	Relational Database Management System.
ORM	Object Relational Mapping.
ORMDBMS	Object Relational Mapping Database Management System.
FIP-UMMU	Fakultas Ilmu Pendidikan Universitas Muhammadiyah Maluku Utara.
RTM	Requirements Traceability Matrix
MVP	Minimum Viable Product
CORS	Cross-Origin Resource Sharing
IETF	Internet Engineering Task Force
CRUD	Create, Read, Update, Delete
SDK	Software Development Kit
DOM	Document Object Model
FSM	Finite State Machine
JSONB	JavaScript Object Notation Binary
CI/CD	Continuous Integration/Continuous Delivery
YAML	Yet Another Markup Language
UAT	User Acceptance Testing
CLI	Command Line Interface

Chapter 1 Introduction

The Student Talent Development Center (STDC) at the University of Kurdistan Hewlêr plays a crucial role in supporting the academic and personal development of students through various services, such as Peer Assisted Learning (PAL). Topping & Ehly (1998) define Peer Assisted Learning (PAL) as "people from similar social groups who are not professional teachers helping each other to learn and learning themselves by teaching". The STDC is able to encourage the development of student skills and abilities by creating a community of learners who work together toward common goals.

In order to enhance the efficiency and effectiveness of its operations, the STDC is seeking to implement a scheduling, feedback, and resource management system that can streamline its workflow. Course timetabling is a complex problem that involves assigning students, teachers, courses, and classrooms to specific times and locations Carter & Laporte (1998). Many universities have turned to virtual learning platforms, such as Moodle, to facilitate this process Al-Ajlan & Zedan (2008). However, these platforms may not always meet the specific needs and requirements of a given institution. As Alvarez-Valdes et al. (2002) note, the format of schedules can vary significantly from one university to another, and it is important to consider the particular constraints and objectives of the course scheduling process.

Course Scheduling, Class-Teacher Scheduling, Student Scheduling, Teacher Assignment, and Classroom Assignment are all few of the subproblems described by Carter & Laporte (1998) in their assessment. It is necessary to consider a variety of constraints and needs in order to solve any of these smaller problems. The goal of this project is to develop a web-based software system that can address these challenges and optimize the STDC's workflow. The system will allow tutors, students, and administrators to schedule and confirm tutoring sessions more efficiently, and provide a more comprehensive view of each group's progress to the administration. To achieve this goal, it will be necessary to carefully design and implement the system, taking into account the specific needs and requirements of the STDC.

This requires a thorough analysis of the current workflow and the identification of any bottlenecks or inefficiencies. It also involves the development of a database to store and manage the necessary data, as well as the creation of user-friendly interfaces for the various stakeholders.

The proposed system for STDC enhances the efficiency and effectiveness of the center's duties. By automating routine tasks and providing a more comprehensive view of each group's progress, the system helps to optimize the use of resources and support the development of student talent.

In addition to the benefits for the STDC, the implementation of a scheduling, feedback, and resource management system has the potential to contribute to the broader field of education. The demand for efficient and effective solutions to aid in the administration and distribution of education is growing as the pace of technological development accelerates. The development of a web-based software solution to handle the issues that a center like STDC faces has the potential to serve as a model for other institutions looking to better their workflows.

1.1 Problem Statement

Scheduling tutoring sessions for students, maintaining records on tutor availability and attendance, gathering feedback from students on tutors, allocating rooms, and making available tutoring modules visible are all regular parts of the Student Talent Development Center's routine. Beginning with students, they have trouble accessing the provided material in each session, knowing how many modules are available to them, giving feedback on certain sessions, and when and where each session is being held. Then, tutors are unable to indicate their availability for upcoming sessions to students, see the total number of recorded tutoring hours, or monitor session attendance. Consequently, the administration has no way of knowing how many sessions a tutor has had with a particular student. They have trouble allocating rooms for each session, tracking how much each tutor should be paid, and receiving useful feedback from students.

1.2 Objectives

- Automation of UKH Student Talent Development Center's Manual Workflow.

1.3 Thesis Organisation

After Chapter 1 Introduction, which provides an overview of the project and its context, Chapter 2 Background and Literature Review presents a review of relevant literature and research on the topic. Next, Chapter 3 Methodology describes the methodology in which way the project has been developed.

Then, Chapters 4 Analysis and Design covers the planning phase, analysis phase, and design phase of the SDLC with each as section. These sections contain detailed steps taken to plan, analyze, and design the web-based software system. After that, Chapter 5 Implementation and Testing covers the implementation and the testing phase of SDLC including any challenges or issues that were encountered and how they were addressed. The testing phase of the SDLC is covered in the last section of chapter 5, which describes the various testing methods and techniques used to ensure the quality and reliability of the system.

Finally, Chapter 6 Conclusion summarizes the main findings of the project and discusses the contributions of the study. It also provides recommendations for future work and potential areas for further research. Overall, the structure of the thesis reflects a systematic and rigorous approach to the research process, following a logical progression from the introduction to the conclusion.

Chapter 2 Background and Literature Review

Virtual learning platforms, such as Moodle, Google Classroom, and Edmodo, have become increasingly popular in higher education as a means of supporting online and blended learning. These platforms offer a range of tools and features for communication, collaboration, and assessment, and are designed to facilitate the delivery of course content and support student learning.

Several studies have explored the use and effectiveness of virtual learning platforms in higher education, some of these studies will be mentioned briefly while others will be reviewed thoroughly in the following review.

The use of intelligent tutoring systems has been studied way back in 1985 by Anderson et al. (1985), who examined the use of intelligent tutoring systems in education, noting their potential to provide personalized and adaptive instruction. Then, VanLehn (2006) studied the behavior of tutoring systems, identifying factors that can influence their effectiveness, such as the type of task and the level of student engagement. Furthermore, VanLehn (2011) compared the relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems, finding that all three types of tutoring can be effective in supporting student learning.

Additionally, Åström & Murray (2008) examined feedback systems, highlighting the importance of timely and specific feedback in supporting learning. Then, Tadelis (2016) explored the role of reputation and feedback systems in online platform markets, noting their potential to increase trust and encourage participation. Furthermore, Sadik (2017) studied the acceptance of file-sharing systems, such as Google Drive, by students as a tool for sharing course materials.

There are several other types of applications that can be seen between 2008 – 2010 including the Zhou et al. (2010)'s system that was developed for lecture notes searching and sharing, called LESSON, which allows users to easily locate and access notes over the internet. Then, Wang et al. (2010) proposed PeerLearning, a content-based e-learning material-sharing system based on a peer-to-peer network.

In addition to these, Lan et al. (2007) examined the use of a mobile-device-supported PAL system for collaborative early English as a foreign language reading.

Many studies have been done on the many kinds of virtual learning platforms, which may be seen to come in a variety of flavors. These platforms are the subject of a great deal of research. However, in order to determine which of these platforms have the most significant impact and importance to this study, it is necessary to conduct an analysis of platforms such as Moodle, Google Classroom, and Edmodo. This is due to the fact that, among the various virtual learning platforms, these three platforms have come the closest to satisfying the requirements of the Student Talent Development Center.

Starting with Moodle which is a free, open-source learning management system (LMS) that is widely used by institutions around the world. According to Rice & William (2006), Moodle is built on top of a LAMP stack and uses a modular design, which allows users to customize the platform and add additional features through the use of plugins. This flexibility and customizability is a major advantage of Moodle, as institutions can tailor the platform to meet their specific needs and requirements. However, setting up and maintaining Moodle may require a significant amount of time and resources, particularly for larger institutions. Dougiamas et al. (n.d.) notes that Moodle is generally considered to be user-friendly and reliable, with a range of tools and features for communication, collaboration, and assessment.

Kc (2017) conducted a case study of the Moodle-based course management system at Kajaani University of Applied Sciences, finding that the platform was well-received by students and faculty, with a high level of satisfaction reported. However, the study also identified several challenges, including technical issues and a lack of training for faculty. Similarly, Al-Ajlan & Zedan (2008) evaluated the Moodle-based course management system at King Saud University and found that while the platform was well-received, there were also challenges such as a lack of training and technical difficulties.

In terms of efficiency, Moodle offers a range of tools and features that can help in-

stitutions streamline their workflows and improve the delivery of course content. For example, Moodle allows instructors to create and manage online courses, schedule lectures, upload resources and content such as lecture files, and provide feedback to students. It also enables students to access course materials, submit assignments, and collaborate with their peers. Overall, Moodle is a powerful and widely-used LMS that can help institutions support online and blended learning, although its effectiveness may depend on its implementation and the specific needs of the institution.

In contrast with Moodle, Google Classroom is also a free, web-based learning management system (LMS) developed by Google that is built on top of the Google Drive and Google Calendar platforms. According to Iftakhar (2016), Google Classroom is designed to be user-friendly and easy to use, with a simple and intuitive interface that is familiar to many users. Sudarsana et al. (2019) found that the use of Google Classroom in the learning process was beneficial, with students reporting improved organization and communication. However, Mohd Shaharanee et al. (2016) note that there may be limitations to the customizability of Google Classroom, as it is built on top of a specific set of tools and features that cannot be easily modified.

Tadelis (2016) discusses the role of reputation and feedback systems in online platform markets, stating that they can be effective in improving the quality and efficiency of services. Google Classroom includes a range of tools and features for communication, collaboration, and assessment, and is highly scalable, integrating with other Google tools such as Gmail, Docs, and Sheets. Sadik (2017) found that students had a high level of acceptance of file-sharing systems like Google Drive as a tool for sharing course materials. However, it is important for institutions to consider issues of data privacy and security when using cloud-based LMSs like Google Classroom.

In terms of efficiency, Google Classroom is generally considered to be user-friendly and reliable, with a range of tools and features for communication, collaboration, and assessment. However, as noted by Mohd Shaharanee et al. (2016), the effectiveness of the platform may depend on the specific design and implementation of the program, as well as the characteristics of the students and instructors involved. Overall, Google Classroom is a popular LMS that offers a range of benefits for on-

line and blended learning, but it is important for institutions to carefully consider their specific needs and requirements when evaluating its use.

Just like both of the above platforms, Edmodo is also a cloud-based learning management system (LMS) that is designed to be user-friendly and easy to use, with a simple and intuitive interface. According to Dewi (2014), Edmodo is a social learning platform that can be used for blended learning in higher education. The platform includes a range of tools and features for communication, collaboration, and assessment, and is focused on education and the integration of various educational tools and resources. Edmodo is highly scalable and has a strong emphasis on security and data privacy per Gay & Sofyan (2017). However, it may not be as customizable as some other LMSs, and some users may find it less feature-rich compared to other platforms such as Moodle or Google Classroom.

Gay & Sofyan (2017) conducted a study to evaluate the effectiveness of using Edmodo in enhancing student outcomes in an advanced writing course at FIP-UMMU. The authors found that Edmodo was well-received by students and facilitated collaborative learning. However, they also identified some challenges, including a lack of technical support and a need for more training for faculty.

In terms of scheduling, Edmodo includes a calendar feature that allows users to schedule events, assignments, and assessments. Edmodo cannot be used to schedule lectures or other large-scale events. However, the platform does allow users to upload and share resources such as lecture files, and it includes features for providing feedback on lectures and other course materials. Overall, Edmodo is a user-friendly and reliable LMS that can support communication, collaboration, and assessment in higher education.

The above review of the literature on virtual learning platforms in higher education reveals that several studies have explored the use and effectiveness of platforms such as Moodle, Google Classroom, and Edmodo. However, there is a lack of research and project on the use of web applications specifically for Tutoring Centers such as the University of Kurdistan Hewler's Student Talent Development Center in

the Kurdistan Region of Iraq (KRI).

Additionally, much of the existing research has focused on the use of these platforms for course management and delivery for universities, rather than specifically for a tutoring center. This suggests a need for research or a project on the effectiveness of web applications for supporting and enhancing tutoring centers in KRI.

Another reason to consider the development of a new web application is the potential for increased efficiency and effectiveness. A custom web application could be designed to specifically meet the needs and requirements of the university's talent development center, potentially leading to a more streamlined and efficient process. Additionally, a custom web application could offer a range of features and tools that are not available on existing platforms, such as session registration, real-time feedback, and specialized resources and materials.

In summary, the proposed system aims to automate the manual workflow of the tutoring center. By implementing a student talent development center web application for the University of Kurdistan Hewlêr, the project will be the first of its kind in the Kurdistan Region of Iraq (KRI) and it will answer the question of whether such a web application is necessary and, if so, how it can be designed and implemented effectively.

Chapter 3 Methodology

Methods for developing software are systematic approaches to completing a software project. Approaches that are both effective and practical often combine a set of clearly defined stages with a conceptual approach to design or process. There are many different methods to go about developing software. As much as one might wish otherwise, there is a profound lesson to be learned from all available approaches. Before settling on a certain approach, it's wise to be familiarized with the most popular software development approaches. The most well-known software development approaches are the Waterfall Approach, the Iterative Approach, and the Incremental Approach. In the following, I briefly discuss each method before presenting the methodology used for this project.

The traditional waterfall method is a method that follows a linear and sequential progression and divides the software development process into a number of discrete stages in a chronological manner. Planning, analysis, design, implementation, testing, deployment, and maintenance are the steps that make up the traditional waterfall approach. The most significant benefit of using this method is that it makes it possible to have a well-defined and well-organized process of development, where each phase builds upon the one that came before it. However, a potential downside is that once a phase is over, there is little room for iteration or change.

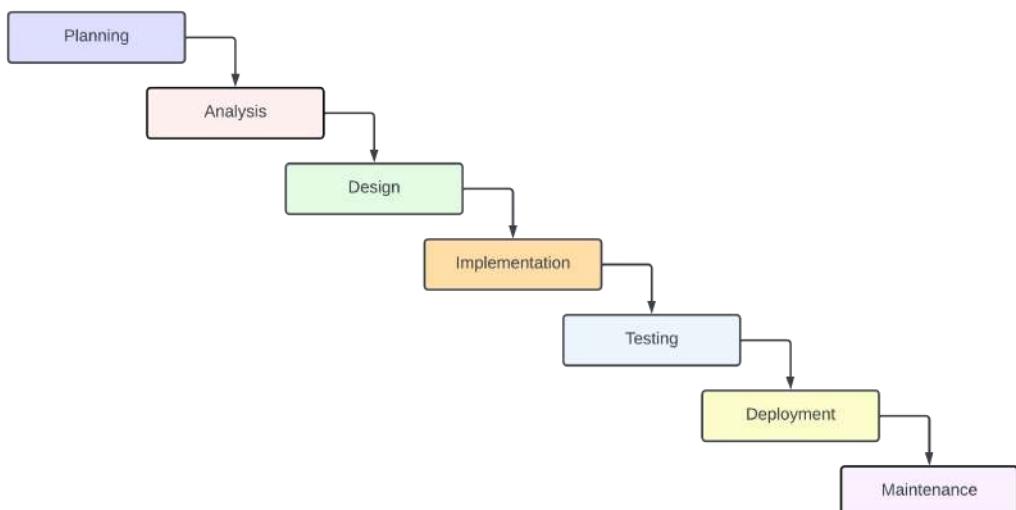


Figure 3.1: Waterfall approach

Approaches to the development of software that are iterative entail the repetition of cycles, sometimes known as iterations, of planning, implementing, and evaluating. As a result, this enables a larger degree of flexibility, as well as the chance to make modifications and enhancements while the product is still being developed. However, due to the fact that the scope of the project may undergo major changes throughout the course of its duration, it may be more challenging to manage and control.

The method of developing software incrementally entails slicing the whole process of development into smaller pieces, which are referred to as increments, and providing a version of the system that is operational at the completion of each iteration. Because of this, a strategy that is more flexible and iterative is possible, since each increment may be tweaked and improved upon depending on the comments and suggestions of users. Nevertheless, it may be more difficult to manage and organize, particularly if the project involves a large number of increments.

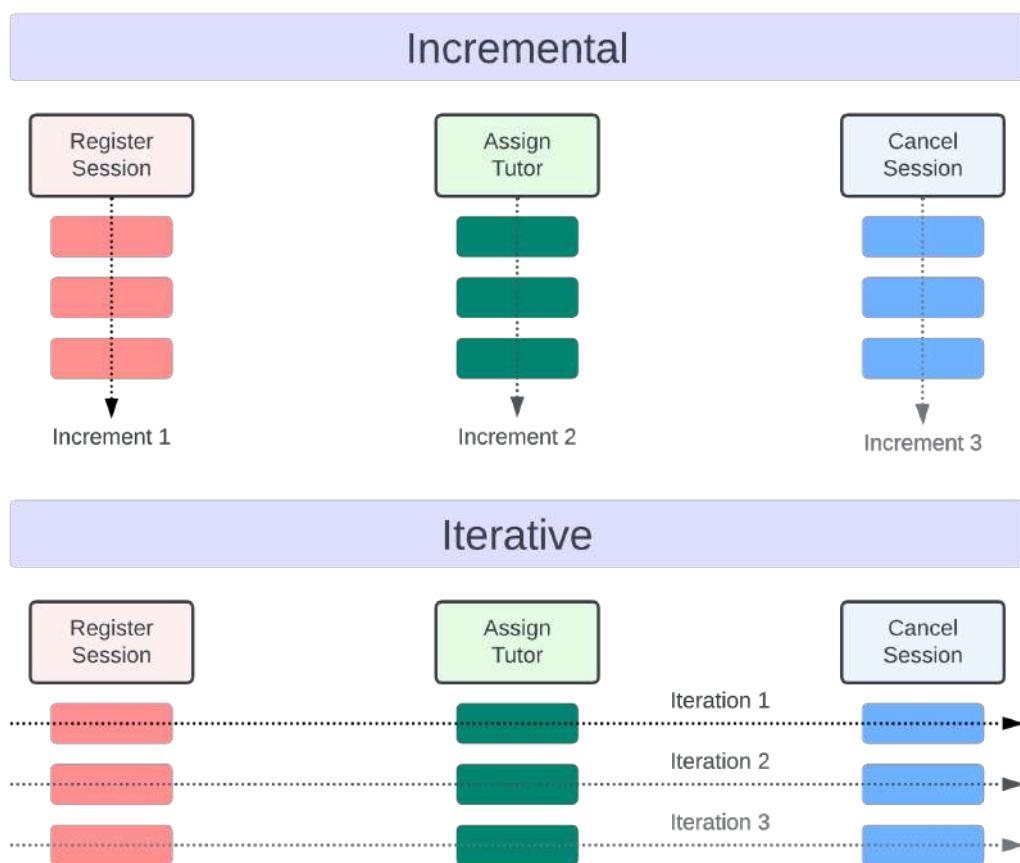


Figure 3.2: Difference between incremental and iterative

The methodology for developing this application is based on a customized version of the waterfall model. I follow the standard stages of the waterfall model for this project; however, I have made certain adjustments so that they are more closely aligned with the particular needs and requirements of this project. The adjustments are including the followings:

1. Deployment and maintenance stages of the traditional waterfall are omitted based on consultation with my supervisor.
2. Testing stage does not start after implementation rather it will start as soon as the requirements are gathered and continues throughout the project.

Planning, analysis, design, implementation, and testing are the different stages that are included in this customized waterfall model. Therefore, it is important to highlight what each stage includes, below here, each stage is explained.

As the initial stage in the process of developing the project, feasibility studies are being carried out. The purpose of these studies is to investigate the practicability of the project that is going to be carried out in terms of its technical, economic, and cultural feasibility. The following stage is the planning phase, which comes at the beginning of the software development life cycle (SDLC) and is hence the first phase. In this phase, I define a thorough project plan that describes the tasks, resources, and timetable for the development process. In addition, I define the stakeholders and risks analysis of the project.

Following the planning phase, I conduct the analysis phase. During this phase, I collect the requirements for the application, which include the needs and expectations of the stakeholders, and document those requirements by creating use cases and activity diagrams. The application's functionality, performance, and overall quality are all determined by the requirements; thus, I treat them as the application's premise. In order to guarantee that the application will satisfy the criteria of the stakeholders, I stay in contact with the stakeholders to validate all of the requirements at the end of this stage.

As mentioned above, instead of delaying the start of testing until the later phases of the development process, I start the testing stage during the analysis phase. This enables me to validate the completeness and correctness of the requirements as soon as they are gathered, and it also enables me to identify and rectify any concerns early on in the process. The functional requirements of the application are the primary focus of the testing activities that take place during the analysis phase. These activities include testing methods such as requirements specification testing, finite state machines, and requirements review.

Following the analysis phase, I conduct the design phase, during which I developed the logical and physical designs for the application. I also take the requirements and turn them into a thorough blueprint for the application. This blueprint includes the architecture of the program as well as its modules, interfaces, and data structures.

After that I start the implementation phase, I develop the system as a whole as well as combine its many individual components. I use an incremental approach as part of the customized waterfall methodology as mentioned above. This means that the development of each component is being done in an incremental way. As a consequence of this, the development of each individual part of the system takes place in its own independent increment. This results in an increment being created for each individual component. I also employ an iterative strategy that is based on Moscow prioritizing across each of these increments.

Table 3.1: MoSCoW prioritization

Priority	Description
Must-Have	Are the essential requirements that must be implemented in order for the application to be usable.
Should-Have	Are the important but not essential requirements.
Could-Have	Are the nice to have but not necessary requirements.
Would-Have	Are requirements that will not be considered in implementations since they are not in the scope.

In each increment, I implement the must-haves first in an iteration manner, followed by the should-haves and then the could-haves. This allows me to deliver a working version of the application at each increment, get feedback from the relevant stakeholders and make improvements as necessary. Figure 3.3 illustrates how implementation would work in a system of 3 components.

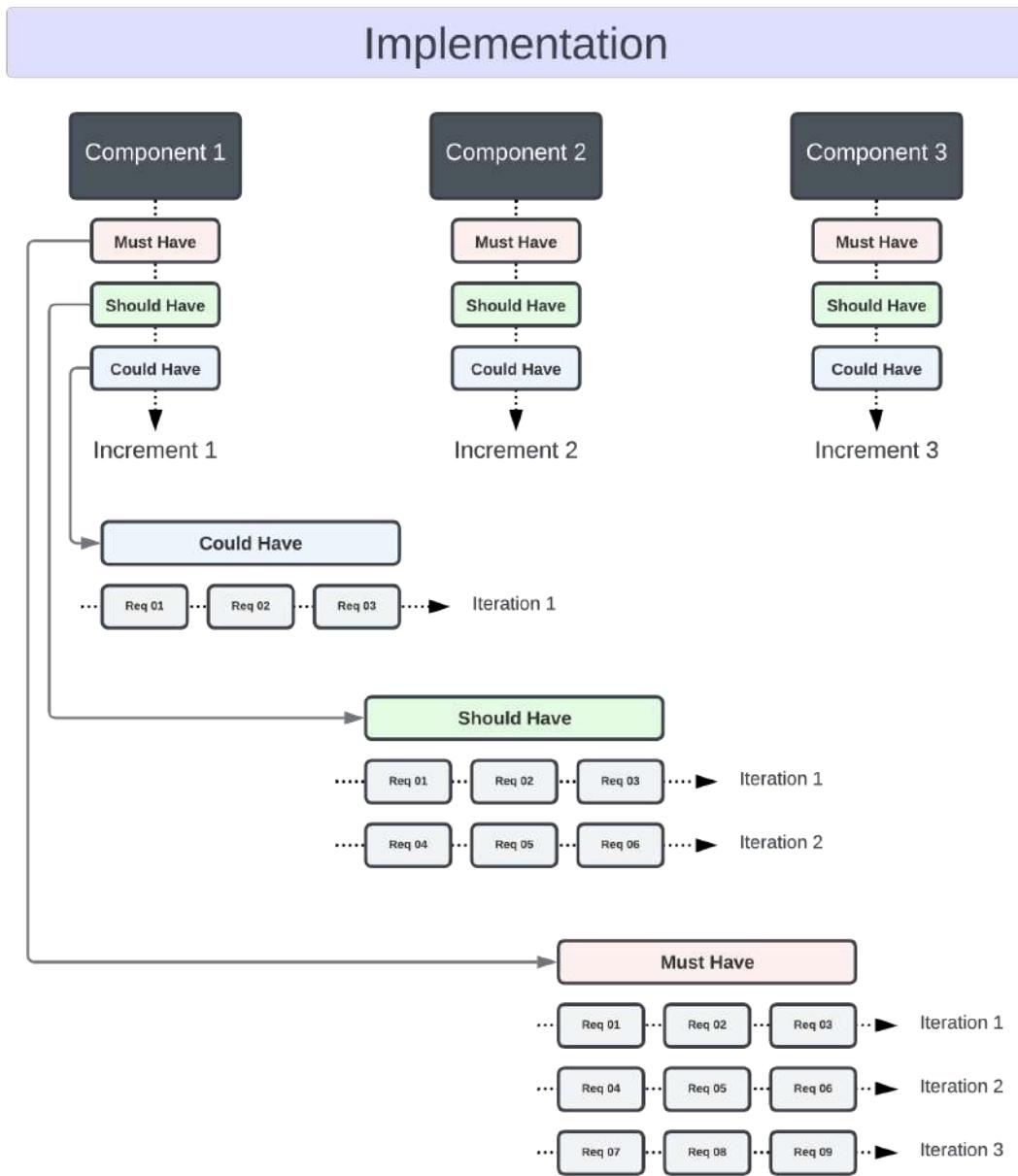


Figure 3.3: Implementation approach

Following the phase of implementation, I conduct the testing phase again, which is a continuation of the testing phase that I started in the analysis up to this point. During

the testing process at this stage, I determine whether or not the application is able to fulfill the criteria and is suitable for the function for which it was designed. Testing is a crucial part of the quality assurance process since it verifies whether or not the program is dependable, effective, and simple to use. Therefore, I conduct a range of tests to examine the application in its entirety. These tests include both functional and non-functional testing, as well as performance testing.

Overall, the customized waterfall development strategy that I use enables me to provide an application of high quality that satisfies the needs and requirements that have been set by the stakeholders. I am able to offer a version of the application that is functional at each stage of the incremental and iterative development process, at which point we may collect feedback and make improvements as necessary. Testing activities ensure that the application is reliable, efficient, and user-friendly. They also identify any issues or improvements that need to be made before the application is handed over.

3.1 Software and Hardware Requirements

It is planned to make use of a wide array of tools, both software, and hardware, in the process of developing this project. Ruby, Ruby on Rails, PostgreSQL, RSpec, React JS, JavaScript, HTML, CSS, Material UI, and Redux are some of the software tools that are available. The application will not only be created using these tools but also tested to guarantee that it is both functional and user-friendly.

In terms of hardware, a laptop or PC, keyboard, mouse, and internet connection will be required. The laptop or PC will be used as the primary development machine, while the keyboard and mouse will be used for input. An internet connection will be necessary to access online resources and connect to remote development tools.

In addition, modeling and documentation applications like Lucid chart, Figma, and LaTeX will be used throughout the course of the project. These tools will be used in the production of high-quality documentation for the project, as well as the creation of diagrams and visual models that will be of assistance throughout the analysis and design phase. Last but not least, in order to facilitate the development of the project,

a wide range of development tools, such as Visual Studio Code, Git, GitHub, Heroku, and Open-API, will be used. These tools will be used for code management, version control, and deployment, as well as making sure that the program can be seamlessly integrated with other systems.

Chapter 4 Analysis and Design

This chapter focuses on the Analysis and Design phase of the software development process. The planning phase, analysis phase, and design phase are all integral parts of this chapter, all of which will be explored in more depth during the course of this chapter. The planning phase establishes the objectives and assesses the risks of the project, while the analysis phase identifies the requirements and constraints of the system, both functional and non-functional. The design phase then takes these requirements and creates a blueprint for the final product. The aim of this chapter is to provide a comprehensive overview of the Analysis and Design phase, highlighting the importance of these phases in the software development process and the critical role they play in the successful completion of the project.

4.1 Planning

The following section will contain the Planning Phase of Software Development Life Cycle of Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System.

4.1.1 Stakeholder Identification

The stakeholders of this system will include the users of the system which are the followings:

- Admin (STDC's Administration/Staff).
- Student (Tutees).
- Tutors

4.1.2 Activity Identification

The followings are the activities of each phase identified during which the development of the system should go through successfully and complete by the end of the project in order to deliver a complete system.

- Planning
 - Identify Stakeholders.
 - Identify Activities.
 - Identify Task Duration.
 - Produce Work Breakdown Structure.
 - Identify Critical Path.
 - Conduct Risk Analysis.
- Analysis
 - Requirement Elicitation (gathering)
 - Requirement Specification and Prioritization
 - Produce Use-Cases.
 - Produce Activity Model.
 - Requirement Validation.
- Design
 - Architectural Design
 - Program Design.
 - Class Model.
 - Sequence Model.
 - Interface Design
 - Database Design
 - Produce EARD.
- Implementation
 - Backend Development.
 - Frontend Development.
- Testing
 - Automated Testing
 - Manual Testing

4.1.3 Task Duration Identification

The following duration are presumed duration needed to finish each task. Each of the specified numbers are in days.

Table 4.1: Activities' duration

Activity	Duration (Days)
Identify Stakeholders	1
Identify Activities	1
Identify Task Duration	1
Produce WBS	1
Identify Critical Path	1
Conduct Risk Analysis	1
Requirement Elicitation	3
Requirement Specification	6
Requirement Validation	3
Architectural Design	1
Program Design	10
Interface Design	7
Database Design	2
Backend Development	28
Frontend Development	28
Automated Testing	21
Manual Testing	32

4.1.4 Work Breakdown Structure

The following tree shows the Work Breakdown Structure of the specified tasks above.

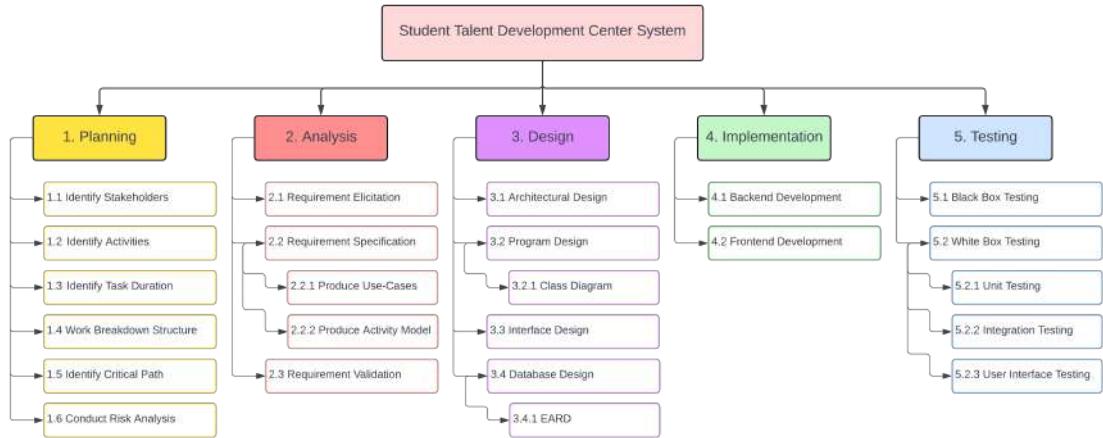


Figure 4.1: Work breakdown structure

4.1.5 Identify Critical Path

The following is the finding of the critical path.

Table 4.2: Activity dependency table

Activity	Predecessor	Duration (Days)
1.1	1.2	1
1.2	-	1
1.3	1.2	1
1.4	1.1, 1.3	1
1.5	1.4	1
1.6	1.5	1
2.1	1.6	3
2.2	2.1	6
2.3	2.2	3
3.1	2.3	1
3.2	3.1	10
3.3	3.2	7
3.4	3.2	2
4.1	3.3, 3.4	28
4.2	4.1	28
5.1	2.1	21
5.2	5.1	32

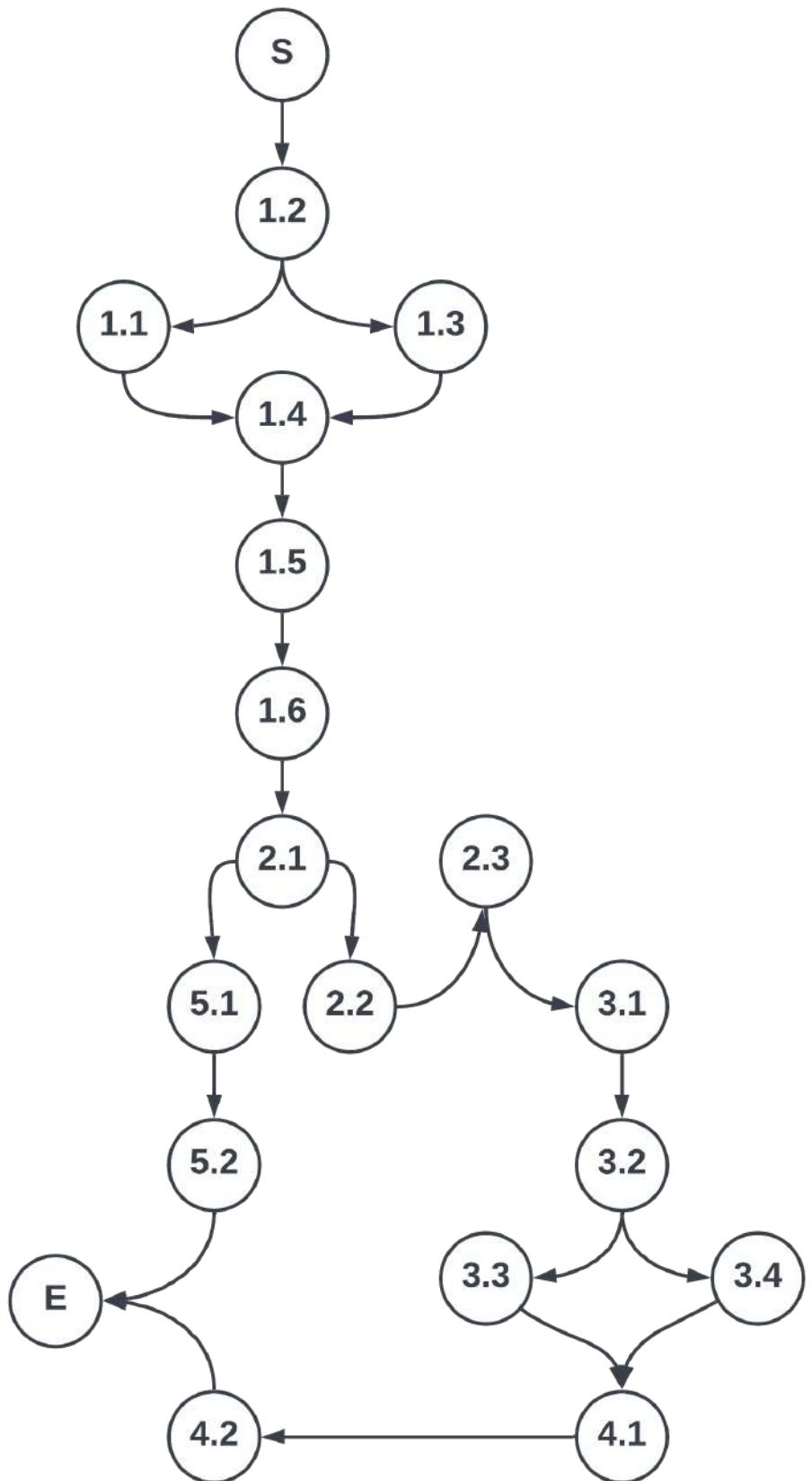


Figure 4.2: Network diagram

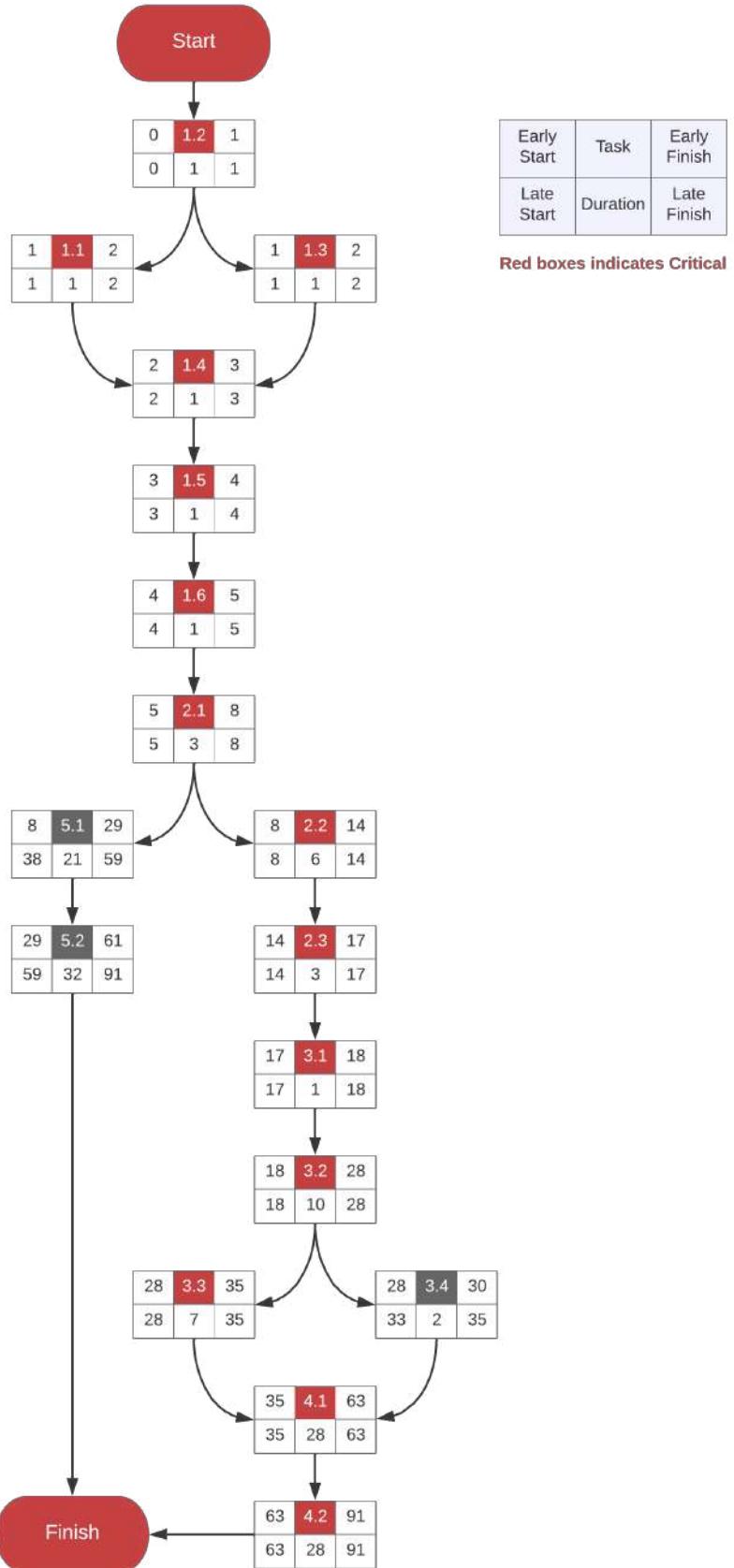


Figure 4.3: Critical path (pert chart)

4.1.6 Risk Analysis

The following is the risk assessment done through analysis of each risk, its likelihood, impact, mitigating strategies, and alternative solution in case of failure.

1. Low Stakeholder Engagement

A lack of stakeholder participation is one of the most frequent software development risks. This occurs due to a lack of communication between project stakeholders and developers, in this case me. In addition, sometimes stakeholders feel disregarded on the project's requirements, resulting in poor participation since they feel excluded/unaware of the project's status.

Likelihood of the risk to happen: HIGH

Impact of the risk when it happens: HIGH

Mitigating Strategy would be to have engaged in the project consistently by updating them regularly on project progress and encouraging them to be involved by confirming the requirements. Having regular meetings with them to keep the communication line with stakeholders can be another strategy to mitigate this risk.

In case of failure, an alternative would be to confront stakeholders and remind them the purpose of the project and that their participation is to improve their lives. Listening to their problems and addressing them during more frequent meetings.

2. Change of Staff or Management at Student Talent Development Center

Although the likelihood of this occurring is low, the possibility still exists. UKH staff and management are constantly replaced. This raises concerns about having different personnel at STDC who may not cooperate or be involved in the project, as well as altering the center's business workflow.

Likelihood of the risk to happen: LOW

Impact of the risk when it happens: MEDIUM

Mitigating Strategy would be to immediately approach new staff/management

upon their arrival at work. Explain to them the objective of the project and the significance of their participation, and have them alert me of any changes to the organization' workflow.

In case of failure, an alternative would be to seek the assistance of former personnel or management in persuading new staff or management of the significance of the project and the need for their participation.

3. Lack of Technical Skills

In software development, technical risks are frequently a setback, something that is not instantly obvious but has severe negative implications. In an attempt to create a unique product, it is common to employ cutting-edge technologies, which might have a number of severe drawbacks and are hard to learn.

Likelihood of the risk to happen: HIGH

Impact of the risk when it happens: HIGH

Mitigating Strategy would be to recognize my weaknesses and learn them before to the implementation phase. Sometimes it is difficult to anticipate these limitations, therefore when a risk develops during implementation, one response would be to seek assistance from colleagues and acknowledge their assistance.

In case of failure, an alternative would be to postpone this functionality temporarily and concentrate on completing the other duties at hand, while conducting research to tackle the issue at hand. This is not always the best option, but it can save time and effort spent on a problem that cannot be solved at the moment.

4. Incorrect Time Estimation

A single day might result in substantial advantages or disadvantages in a project's progress. There are a number of potential causes for time constraints, such as the wrong initial estimation and setting of the project's duration, a shortage of available technical knowledge, or an unanticipated and urgent new requirement.

Likelihood of the risk to happen: MEDIUM

Impact of the risk when it happens: MEDIUM

Mitigating Strategy would be to have extra weeks available so that if a task is delayed, it would not affect the project's deadlines. An alternative would be to work more during a given week to prevent similar situations from occurring.

In case of failure, an alternative would be to take a few days off work and concentrate solely on the project and its tasks. Catch up as quickly as possible.

5. Resource Unavailability from UKH's IT Department (Lack of Support)

Although the possibilities of this happening are not high, nonetheless it is expected not to get support from IT Department of University of Kurdistan Hewlêr, since this project is not done through their supervision.

Likelihood of the risk to happen: LOW

Impact of the risk when it happens: LOW

Mitigating Strategy would consist of constructing the required components and replicating the required data and services. This may take time, but it is the only viable option.

In case of failure, an alternative would be to communicate with STDC administration in an effort to persuade them to assist and support by providing mock versions of their main services.

6. Health Issues (Illness)

Although health concerns can arise at any time for a variety of reasons and are hard to anticipate, it is not difficult to prevent known illnesses that could delay the project by days or weeks. This risk will disrupt the project's progress and cause delays that cannot be afforded. Considering the previous critical path, it is clear that the project must keep to its deadlines, with only a narrow window for delay.

Likelihood of the risk to happen: HIGH

Impact of the risk when it happens: MEDIUM

Mitigating Strategy would be to follow the WHO's guidelines to prevent illness. Follow clean practices, use a mask, and avoid the public if it is not absolutely required. Testing frequently and early to prevent falling behind drastically.

In the case of failure, an alternative would be to contact Registry and submit a mitigating circumstances form in order to extend the project's deadline. Although this is not the best solution, it is the only practical one.

7. Unpredictable External Risks (e.g., Failing a Module)

Unpredictable External Risks is another obstacle that might be confronted during project management. External risks in software development are not so uncommon, risks like introduction of new university laws, changes in the priorities of stakeholders, or even personal issues like failing a module could happen anytime.

Likelihood of the risk to happen: HIGH

Impact of the risk when it happens: LOW

Mitigating Strategy would be to follow up with stakeholders on new priority changes, keep up to university works, and do not procrastinate on studying for this will be hard to mitigate.

In the case of failure, an alternative would be to work in the weekends and holidays to focus on studies and university assignments.

8. Making a lot of Assumptions

When it comes to obscure requirements, it is easy for developers to develop an addiction to making a large number of assumptions. This results in a different product or a product that does not address the issue at hand. And often, starting from scratch is a burden because it is both time-consuming and requires a larger expenditure than allocated. Given that the time frame for this project is extremely tight and does not permit redoing a feature, it is essential that no assumptions be made regarding how things should function.

Likelihood of the risk to happen: HIGH

Impact of the risk when it happens: HIGH

Mitigating Strategy would be to communicate with stakeholders more frequently, to take their ideas into consideration, and to clarify matters that are unclear.

In the case of failure, an alternative would be taking advice from professors who have a good background in systems development and take their opinion on certain matters.

Risk Assessment Findings

The Student Talent Development Center project risk analysis was concluded by examining a variety of hazards across domains. This evaluation identified possible project roadblocks, assessed their likelihood and impact, and developed preventative actions and alternatives if the risk materialized.

This risk assessment acknowledges that every endeavor has risks. The analysis focused on anticipating and mitigating these hazards. Risk management is continuous, nevertheless. As the project continues, the risk environment may change, requiring regular analytical updates. While we've tried to anticipate the biggest dangers, it's important to be flexible to handle the unexpected. To complete the project, these risks must be managed.

4.2 Analysis

The following section will contain the Analysis Phase of the Software Development Life Cycle of Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System.

4.2.1 Requirement Elicitation (Requirements Gathering)

The Student Talent Development Center of the University of Kurdistan Hewlêr offers a variety of services to assist its students in succeeding, including Peer Assisted Learning (PAL), Wellbeing & Counseling, and Personal Academic Tutors. The followings are the requirements gathered from the stakeholders of the project for the center's Peer Assisted Learning subcenter. The below scenario demonstrates what the stakeholders are looking for.

Scenario

As a student at the University of Kurdistan Hewlêr, Sarkazh logs into the Student Talent Development Center (STDC) web application using his UKH email and password. He is immediately presented with a list of courses that are currently being taught in the STDC, with his current courses appearing at the top. He clicks on the course "Introduction to Programming" to view more details about it. He also notices that the upcoming lesson's time and venue are displayed when he views the session. He remembers that he has attended this session before, so he clicks on the "History" button to view past sessions he has attended.

Sarkazh sees that there is an upcoming session for the "Introduction to Programming" course, so he clicks on the "Register" button to sign up for it. He also sees that there are materials uploaded for the sessions he has attended, so he clicks on the "Materials" button to view them.

Meanwhile, Didam, one of the tutors at the STDC, logs into the web application using her UKH email and password. She is responsible for teaching the "Introduction to Programming" course, so she clicks on the "Sessions" button to view the list of sessions she is teaching. She sees that Sarkazh has registered for the upcoming session, so she clicks on the "Students" button to view the list of students who have

registered for it. She also sees that she has to enter the attendance of the previous session, so she clicks on the "Attendance" button to do so.

Didam remembers that she has another class at the same time as the upcoming session, so she clicks on the "Availability" button to update her schedule. She also sees that she has to upload materials for the session, so she clicks on the "Materials" button to upload the files. She also sees that she has already taught for 15 hours, which is displayed on her profile.

Finally, the admin at the STDC, Ranj, logs into the web application using his UKH email and password. He is responsible for managing the courses, sessions, venues and tutors at the STDC. He clicks on the "Courses" button to view the list of courses currently being taught at the STDC. He sees that he needs to create a new course, so he clicks on the "Create" button to do so. He also sees that he needs to update an existing course, so he clicks on the "Update" button to do so. He also sees that he needs to delete an existing course, so he clicks on the "Delete" button to do so.

Ranj also clicks on the "Sessions" button to view the list of sessions currently being held at the STDC. He sees that he needs to create a new session, so he clicks on the "Create" button to do so. He also sees that he needs to update an existing session, so he clicks on the "Update" button to do so. He also sees that he needs to delete an existing session, so he clicks on the "Delete" button to do so.

Ranj also clicks on the "Venues" button to view the list of venues currently available at the STDC. He sees that he needs to create a new venue, so he clicks on the "Create" button to do so. He also sees that he needs to update an existing venue, so he clicks on the "Update" button to do so. He also sees that he needs to delete an existing venue, so he clicks on the "Delete" button to do so.

Finally, Ranj clicks on the "Tutors" button to view the list of tutors currently working at the STDC. He sees that he needs to onboard a new tutor, so he clicks on the "Onboard" button to do so. He also sees that he needs to assign a tutor to a course, so he clicks on the "Assign" button to do so. He also sees that he needs to list the attendance of a session using the specified button. He also clicks on the "Feedback" button to view the student feedbacks and view individual feedback.

Requirement Categorization and Prioritization

Each requirement below is categorized and prioritized base on the following format:

- ***System Component***

- MoSCoW Priority

- **Associated Requirement ID** – Use Case Name – Actor.

- Brief Description.

It is also worth mentioning that the following requirement are the requirements that will be used and referred to as a reference of the system's whole functional requirements. The unique associated requirement ids are used throughout the project to reference a specific requirement.

- ***System Core***

- Must Have

- **RQ01** – Login – All Users.

- All users must be able to login using UKH email and Password.

- **RQ02** – Enter Attendance – Tutor.

- Tutor must be able to enter attendance of each session.

- **RQ03** – List Courses – All Users.

- All users must list courses taught in STDC. (Current User's courses should appear on top).

- **RQ04** – View Course – All Users.

- All users must be able to view an individual course.

- **RQ05** – Create Course – Admin.

- Admin must be able to create a new course.

- **RQ06** – Update Course – Admin.

- Admin must be able to update a course.

- **RQ07** – Delete Course – Admin.

- Admin must be able to delete a course.

- **RQ08** – List Sessions – All Users.
All users must be able to list sessions.
- **RQ09** – View Session – All Users.
All users must be able to view an individual session.
- **RQ10** – Create Session – Admin.
Admin must be able to create a new session.
- **RQ11** – Update Session – Admin.
Admin must be able to update an existing session.
- **RQ12** – Delete Session – Admin.
Admin must be able to delete an existing session.
- **RQ13** – List Venues – Admin.
Admin must be able to list venues.
- **RQ14** – View Venue – All Users.
All users must be able to view an individual venue.
- **RQ15** – Create Venue – Admin.
Admin must be able to create (register) a new venue.
- **RQ16** – Update Venue – Admin.
Admin must be able to update an existing venue.
- **RQ17** – Delete Venue – Admin.
Admin must be able to delete an existing venue.
- **RQ18** – Onboard Tutor – Admin.
Admin must be able to onboard tutors.
- **RQ19** – Assign Tutor – Admin.
Admin must be able to assign tutors courses.
 - Should Have
 - **RQ20** – View Attendance – Admin.
Admin and Tutor should be able to list the attendance of a session

using specified button.

- **RQ32** – View Profile – All Users.

All users should be able to view their profile and its details.

- ***Scheduling System***

- Must Have

- **RQ21** – View Time and Venue – All Users.

All users must see upcoming lesson's time and venue when they view a session.

- **RQ22** – Assign Time and Venue – Admin.

Admin must be able to assign time and venue to each session.

- Should Have

- **RQ23** – View History – Student & Tutor.

Student and Tutor should see the history of sessions they have attended.

- **RQ24** – Enter Availability – Tutor.

Tutor should be able to enter their availability.

- **RQ33** – Register for Session – Student.

Students should be able to register for upcoming session.

- **RQ34** – View Registered Students – Tutor.

Tutor should be able to view students who have registered for upcoming session.

- Could Have

- **RQ25** – Cancel Session – Tutor.

Tutor could cancel a session.

- **RQ26** – View Hours – Admin & Tutor.

Admin and Tutor could see how many hours tutor have tutored when viewing tutor profile.

- ***Feedback System***

- Should Have

- **RQ27** – Provide Feedback – Student.

Student should be able to provide feedback after each session.

- **RQ28** – View Feedbacks – Admin.

Admin should be able to list student feedbacks.

- **RQ31** – View Feedback – Admin.

Admin should be able to view student feedback individually.

- **RQ35** – List Questions – Admin.

Admin must be able to list questions.

- **RQ36** – View Question – All Users.

All users must be able to view an individual question.

- **RQ37** – Create Question – Admin.

Admin must be able to create (register) a new question.

- **RQ38** – Delete Question – Admin.

Admin must be able to delete an existing question.

- ***Resources System***

- Should Have

- **RQ29** – Access Material – All Users.

All users should be able to access materials uploaded.

- **RQ30** – Upload Material – Tutor.

Tutors should be able to upload files (material) for a given session of a course.

Further Constraints to Functional Requirements

- Students won't be able to provide feedback for sessions older than a week.
- Students won't be able to provide feedback to sessions they have not participated in.
- Students must see the material uploaded only for sessions that they have attended.
- Tutors upload files to sessions which belongs to a course.

Non-Functional Requirements

- Performance
 - The ability to have concurrent users on our web application.
 - Each request should be processed within no more than 5 seconds.
- Security
 - To keep the system secure, standard encryption methods such as RS256 will be used.
 - Implementing OAuth 2.0 and OpenID connect to keep the system secure.
 - Using Json Web Token (JWT) to keep sessions and to have secure up to date security standards.
- Reliability
 - System can fail no more than 3 hours per year.
- Usability
 - The users can easily determine what a feature is and what it can do. E.g. clicking on a button with an icon of a magnifier should open search bar.

4.2.2 Requirement Specification

The following is the Analysis phase's Requirement Specifications document. In this phase, requirements are analyzed using Use Cases and Activity Models. The analysis is done using the following tools:

1. UML Use Case Model
2. Use Case Tabular Form
3. UML Activity Model

Four elements compose the Student Talent Development Center Web Application. The System Core, Scheduling System, Feedback System, and Resource Management System. These four components are formed based on the requirements above.

Microsoft Azure Business-to-Consumer (B2C) Identity Server interacts with the System despite the fact that it has four internal components. This external component is integrated but not a part of the system. It provides security and abstraction to user management concerns.

UML Use Case Model

The following are the use case models of each component of the system.

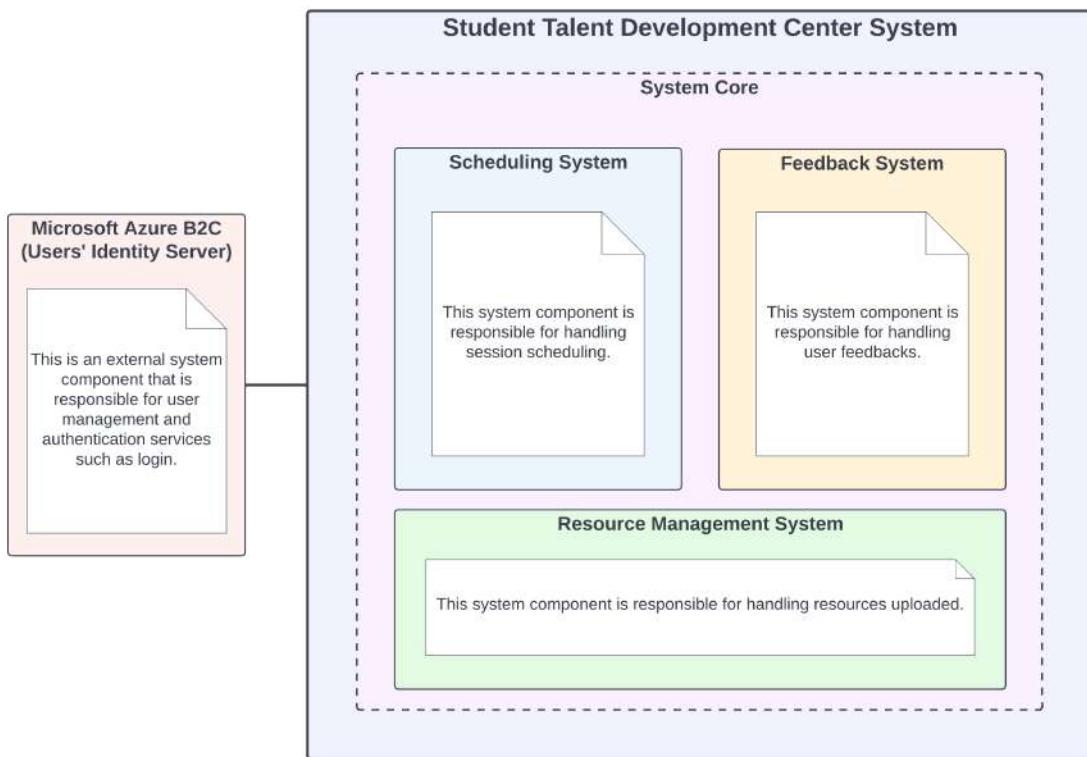


Figure 4.4: Use case model - conceptual system overview

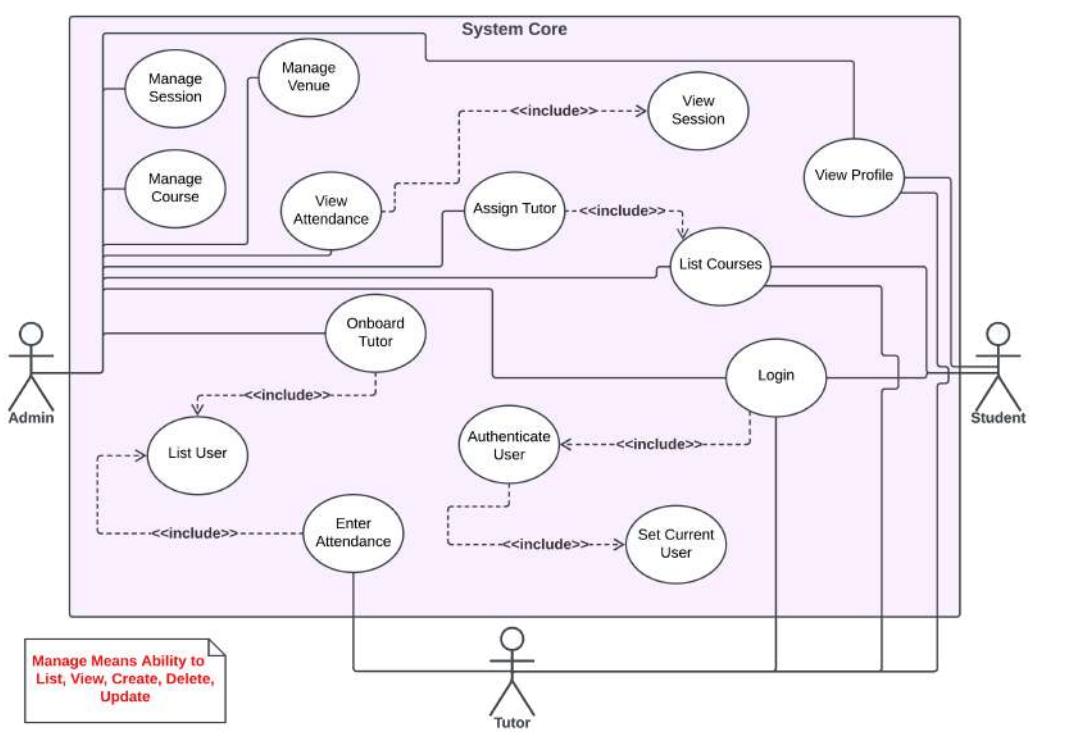


Figure 4.5: Use case model - system core

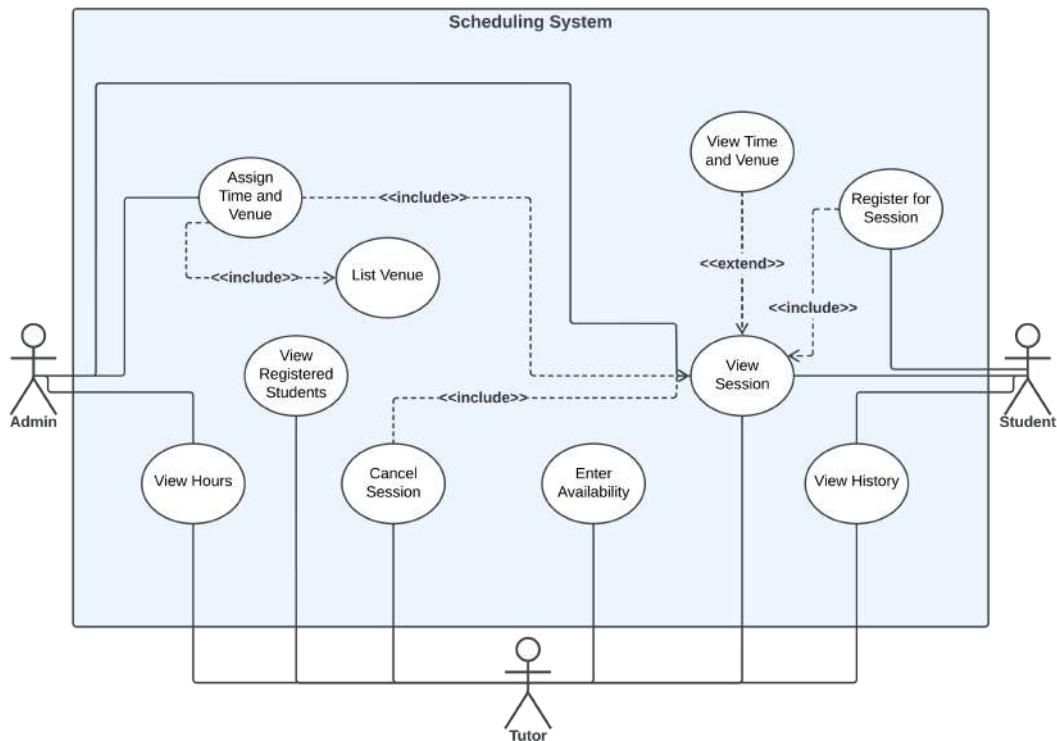


Figure 4.6: Use case model - scheduling system

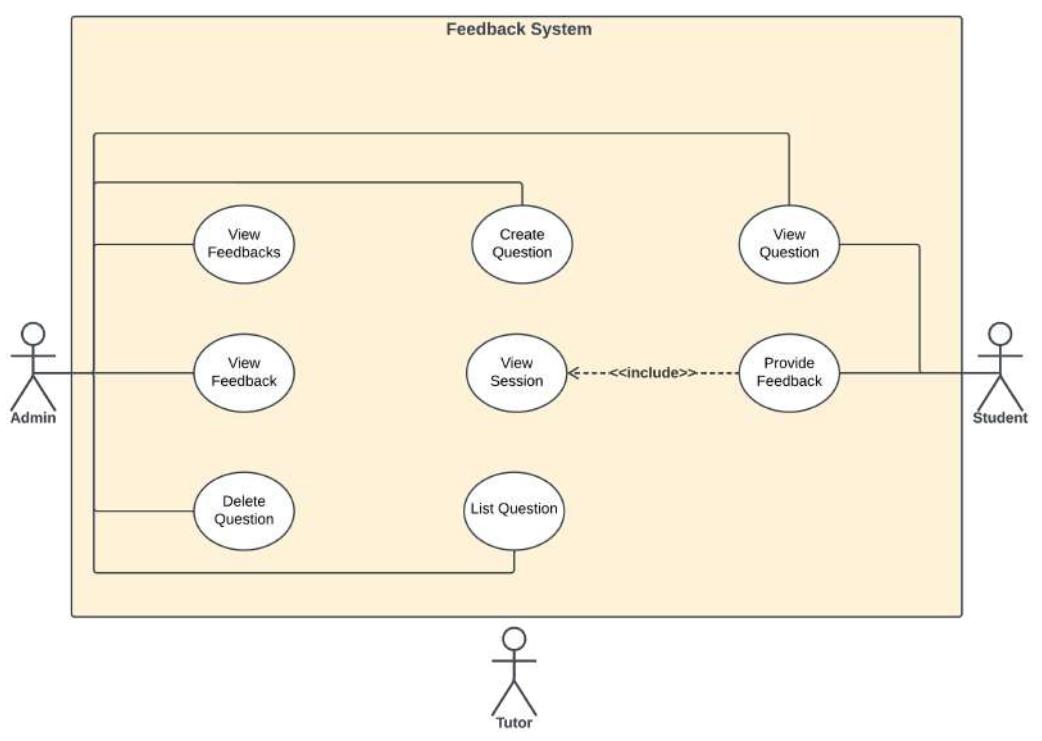


Figure 4.7: Use case model - feedback system

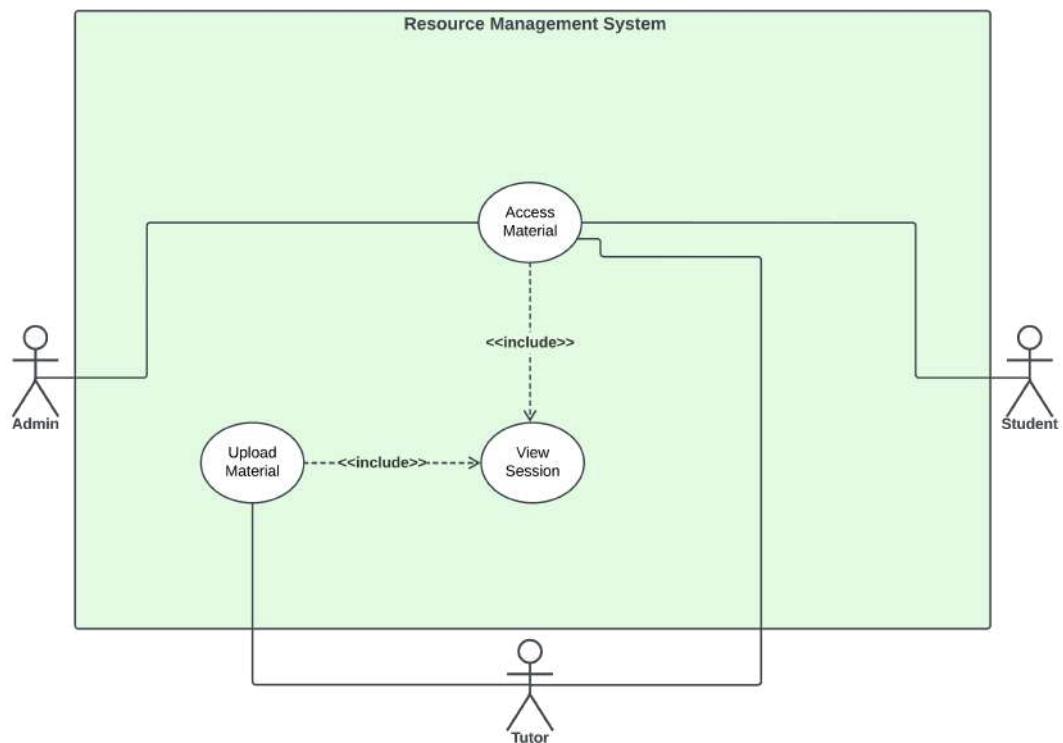


Figure 4.8: Use case model - resource management system

Use Case Tabular Form

The use case tabular form is a detailed version of the above UML Use Cases. Each use case is explained in detail with specified steps, preconditions, post conditions, input, trigger, and so on. Below here the table entries are explain and one example of these use cases are provided, however the rest can be accessed in the Appendix A1.

Table 4.3: Use case tabular form – fields' description

Use Case Field	Description
Use Case ID	Unique ID associated with the use case.
Version	Last updated version.
Last Updated	Date of the last modification to the use case.
Name	Name of the use case.
Traceability to	Related documents that are associated with the use case.
Actor	All users who initiate and participate in the use case.
Preconditions	Constraints that must be met for the use case to be taken.
Trigger	Event that initiates the use case.
Input	List of the data input by the user.
Steps	Steps needed to be taken to reach the use case.
Postconditions	Constraints that must be met for the use case to be done.
Exception Flow	Unsuccessful ways this use case might end.
Expected Output	The expected successful outcome of this use case.

Use Case ID	UC01		
Version	2.0	Last Updated	07 Dec 2022
Name	Login		
Traceability to	RQ01, UCM2	Actor	All Users
Preconditions	User should have a valid Microsoft account (UKH Account)		
Trigger	When user opens the application.		
Input	Valid Access Token from Microsoft.		
Steps	<ol style="list-style-type: none"> 1. Click on Login Button. 2. Sign in using Microsoft Account (UKH Account). 3. Grant permissions to the application. 		
Postconditions	<p>User is authenticated. Current User is set. User is created successfully (if not in the database). User is redirected to home page.</p>		
Exception Flow	<ol style="list-style-type: none"> 1. Error message stating “Unauthorized to sign into the application” 		
Expected Output	User is logged into the application and is the current user.		

UML Activity Model

The followings are the activity models of one of each component of the system.

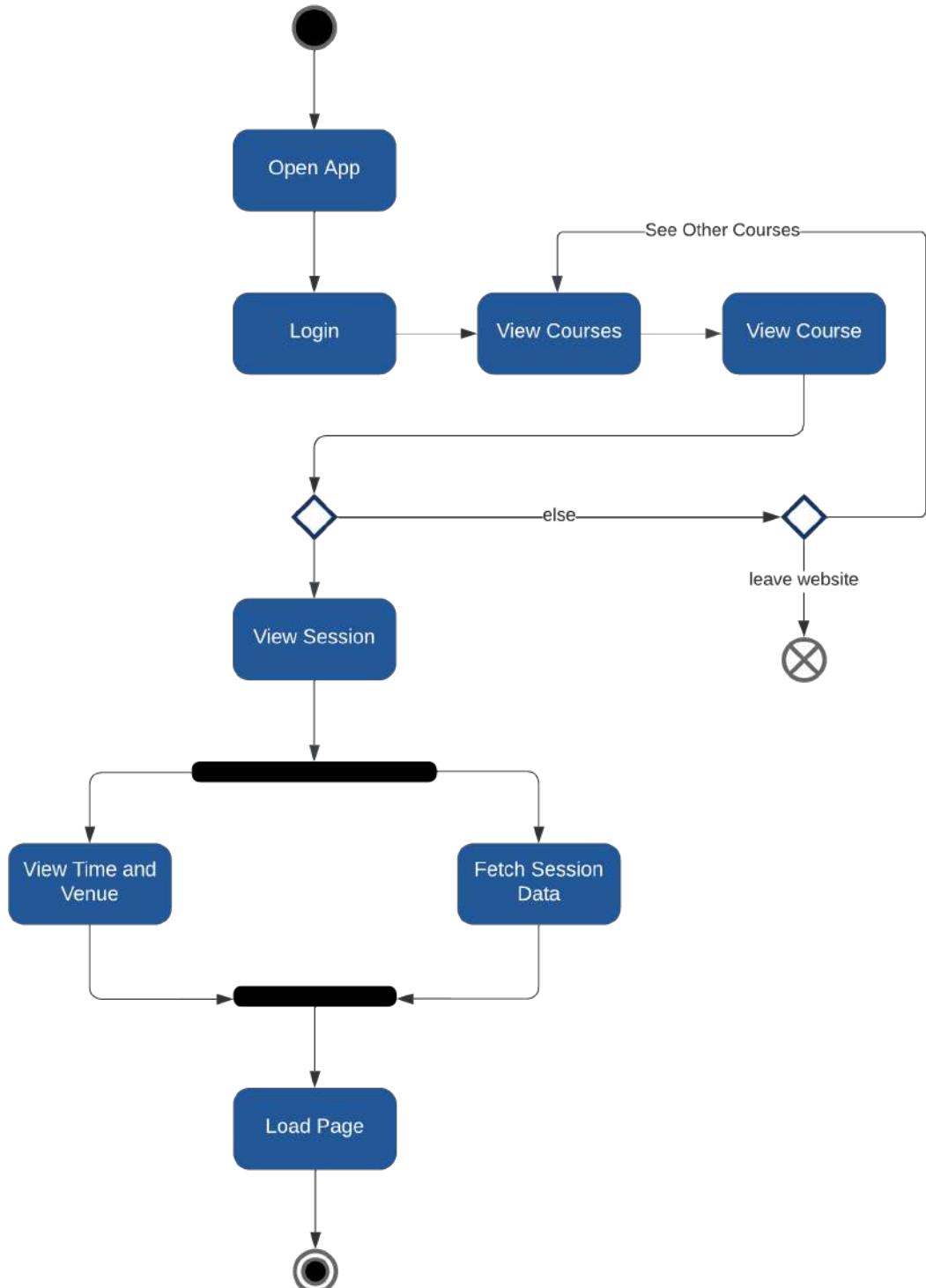


Figure 4.9: Activity model - view session

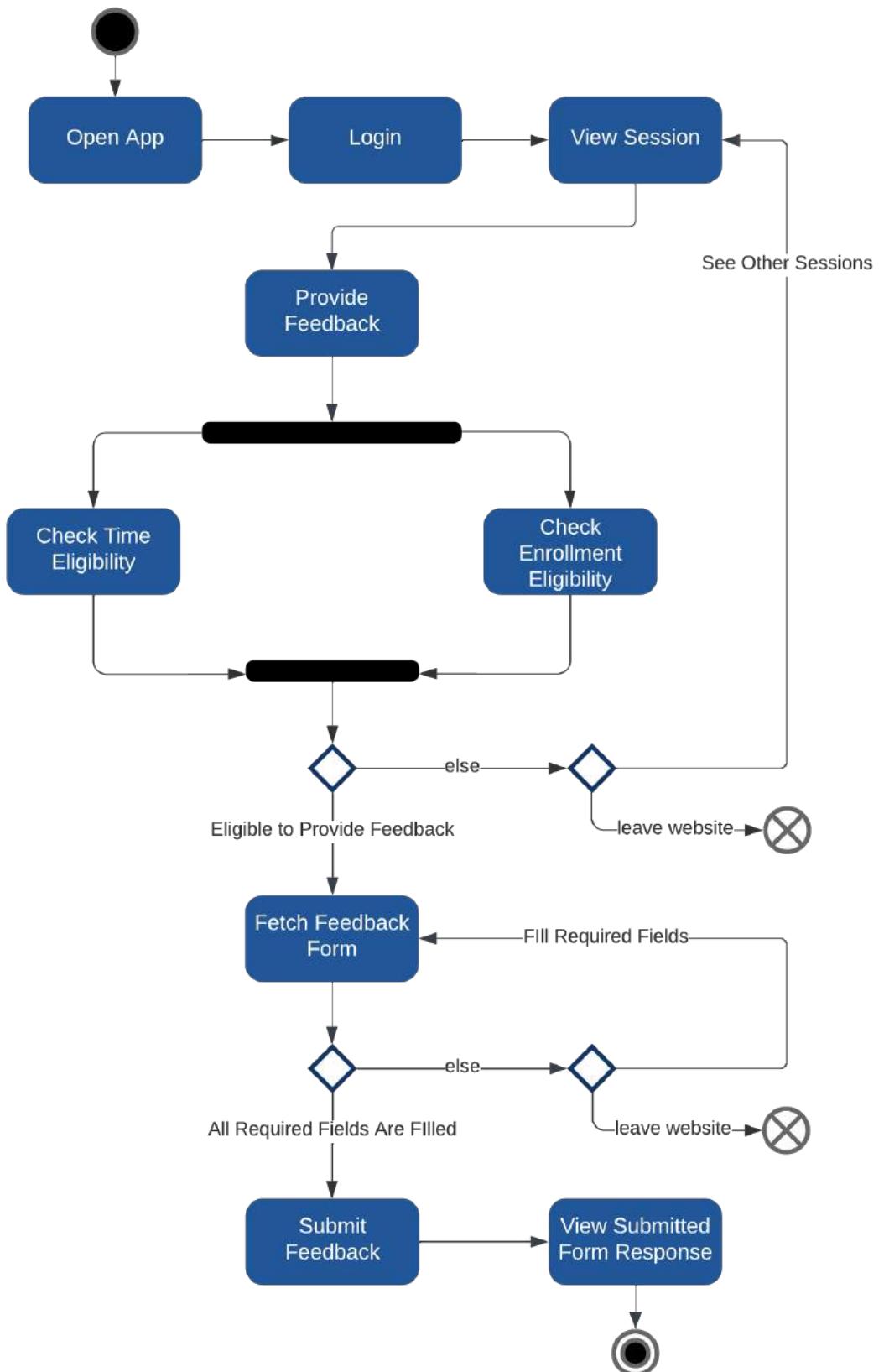


Figure 4.10: Activity model - provide feedback

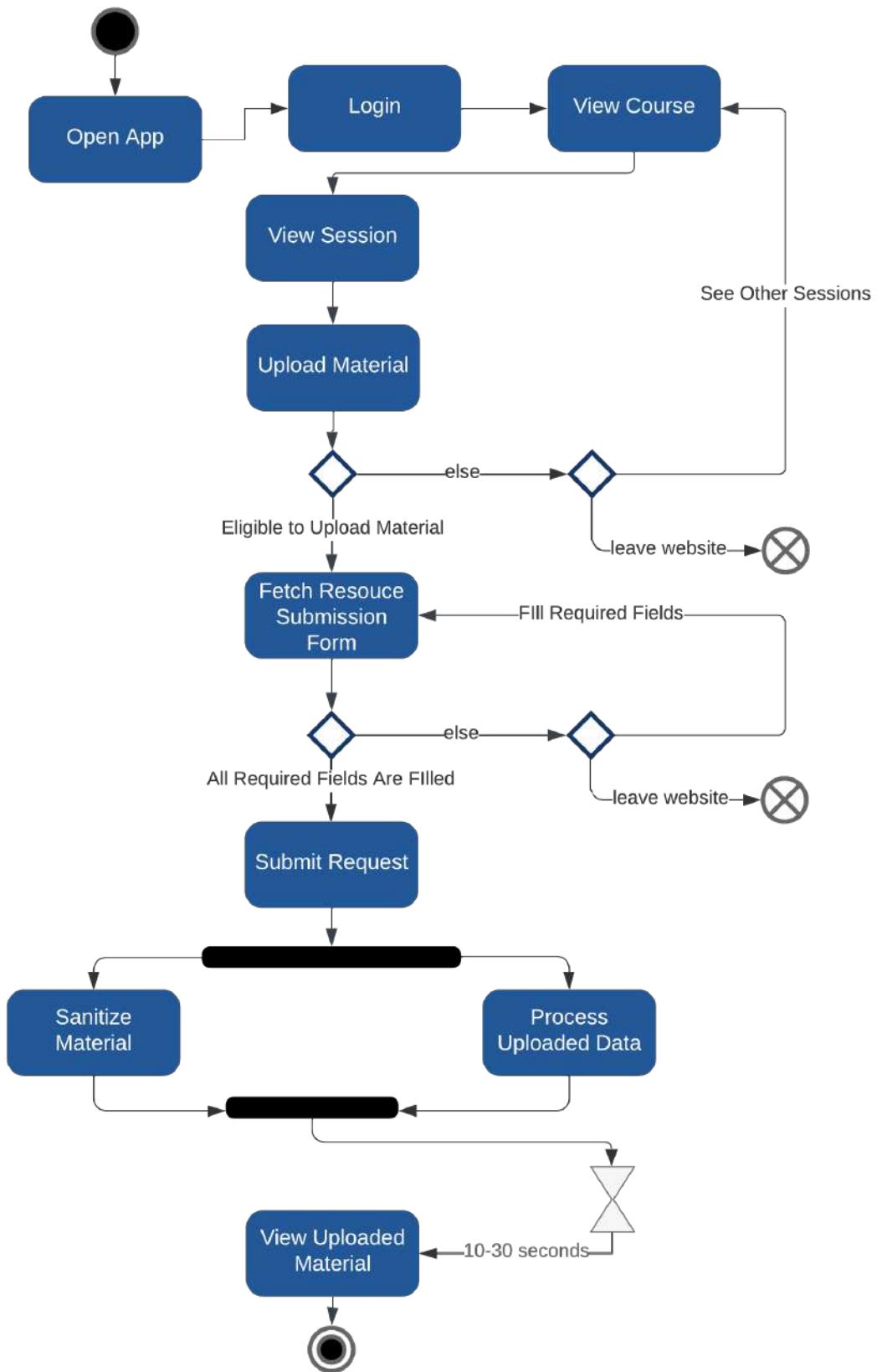


Figure 4.11: Use case model - upload material

4.2.3 Requirement Validation

The requirements validation phase is an essential step in ensuring that this software meets the needs of the end-user at STDC. This section provides an overview of the project's requirements validation procedure.

A list of the validated requirements can be found in the first section of this chapter. The list provides a high level overview of the requirements and their relationship to each other. The requirements have been validated through a combination of methods, including stakeholder meetings, inspection of existing documentation, requirements specification testing (as outlined in chapter 5, section of testing), and finite state machine analysis (also outlined in chapter 5, section of testing).

During the validation process, a number of issues and discrepancies were identified. These are shown in Figures 5.123, 5.9, and Appendices A43, which provide a detailed look at each of the issues and the steps taken to resolve them. While many of the issues required changes to the requirements, some of them have been addressed through the design and development of the system.

The overall effectiveness of the validation process has been positive, and the requirements have been deemed to be of high quality. The validation process has provided valuable feedback that has been used to improve the overall design of the system, and it has helped to ensure that the software will meet the needs of the stakeholders.

In terms of sign-off, stakeholders have been involved throughout the development process and have provided feedback and input on the requirements. While a formal sign-off is not available, their active participation and engagement in the process serves as validation and acceptance of the requirements.

4.3 Design

The following section will contain the Design Phase of the Software Development Life Cycle of Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System.

4.3.1 Architecture Design

The following paper is the Architecture Design of the Student Talent Development Center Web Application. The approach and architecture of developing the application is explained in details below here.

Overview

The four main parts of the Student Talent Development Center Web Application are the System Core, the Feedback System, the Resources Management System, and the Scheduling System. The System also interacts with Microsoft Azure Active Directory B2C which is the Users' Identity Server used for authenticating users and retrieving their information. Student Talent Development Center has a monolithic architecture pattern. Before we explain each part, let's differentiate between the design components.

Physical Architecture

This is the physical separation of the application. Meaning physically separating the application on machine level. Three Tier Architecture is chosen for this.

1. Presentation Tier: Runs on client machine (client's laptop) browser.
2. Logic Tier: Runs on UKH servers. (On Heroku as of development).
3. Database Tier: Runs on UKH Database Vendor's machine. (On AWS as of development).

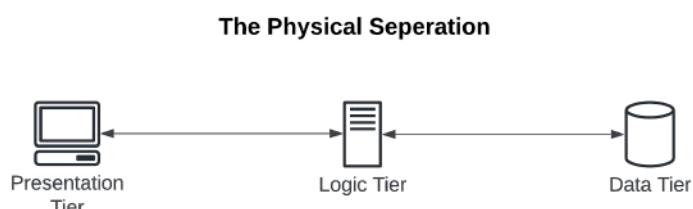


Figure 4.12: Physical separation

Logical Architecture

This is the logical separation of the application. Meaning logically where the application components are. The application has a Client (Frontend) and a Server (Backend).

1. The Frontend: is responsible for presentation tier of the application. Uses different technologies to illustrate user interface.
2. The Backend: is responsible for logic tier and database tier of the application. Application logic and data access logic are all here.

Software Design Pattern

This is the software design pattern, meaning how the application is developed (is coded), this has nothing to do with physical design or logical design. MVC Design Pattern is used.

1. Model: is responsible for business logic, and model classes are here.
2. Controller: is responsible for connecting the bridge between the model and the view. It receives user input.
3. View: is responsible for representing the application user interface.

Implementation Paradigms

This is the approach used to code each component of the logical architecture. Two approaches are used here:

1. Object Orient Programming: is used for the backend of the application
2. Functional Programming: is used for the frontend of the application.

Physical Architecture Design

The physical separation of concerns for the system physically, meaning this separation is machine separation. The machines used as mentioned before will be into 3 tiers. Below here each tier is explained.

1. Presentation Tier

The presentation tier is responsible for showing the user interface of the application to the user. It runs separately from the other layers on clients' machine. The presentation tier is also responsible for communicating with the logic tier.

2. Logic Tier

Processing data, carrying out calculations, and making logical decisions are the tasks that are within the scope of the logic tier. Because practically all of the system's business logic is maintained in this tier, it is sometimes referred to as the "heart" of the application. This tier is deployed onto a dedicated server as its machine that is available at all times.

3. Data Tier

This tier is responsible for handling the system's data. A software that manages read and write access to a database is included inside the data tier or database tier of the system. This tier, which is also known to as a storage tier is hosted on a database server, a different machine from the other tiers.

In a three-tiered system, the logic tier is the point at which all interaction is routed. Both the presentation tier and the database tier are separated from each other and are unable to connect to each other explicitly.

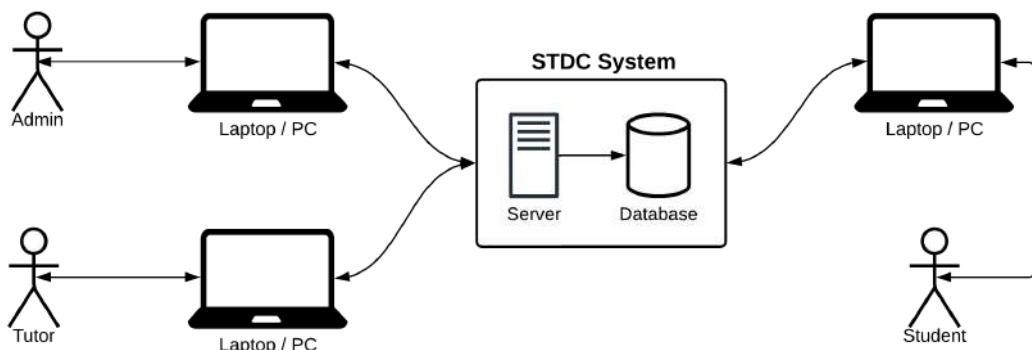


Figure 4.13: User interaction

Logical Architecture Design

The logical design architecture is extremely straightforward and basic. It consists of a client, also known as the frontend, submitting a request to the server, also known as the backend, in order to receive a response from the server. The request coming from the client may be any one of post, get, put, delete, or patch, but it would still need to include an authorization parameter in the header. This authorization parameter would include the user's access token that was acquired from the identity server.

The client's request is then dealt with by the server, and the results are sent back to the client in JSON format. Take note that the server will interact with the database in the event that this becomes necessary. Below here the pattern of development of each the client (frontend) and the server (backend) will be discussed in further details.

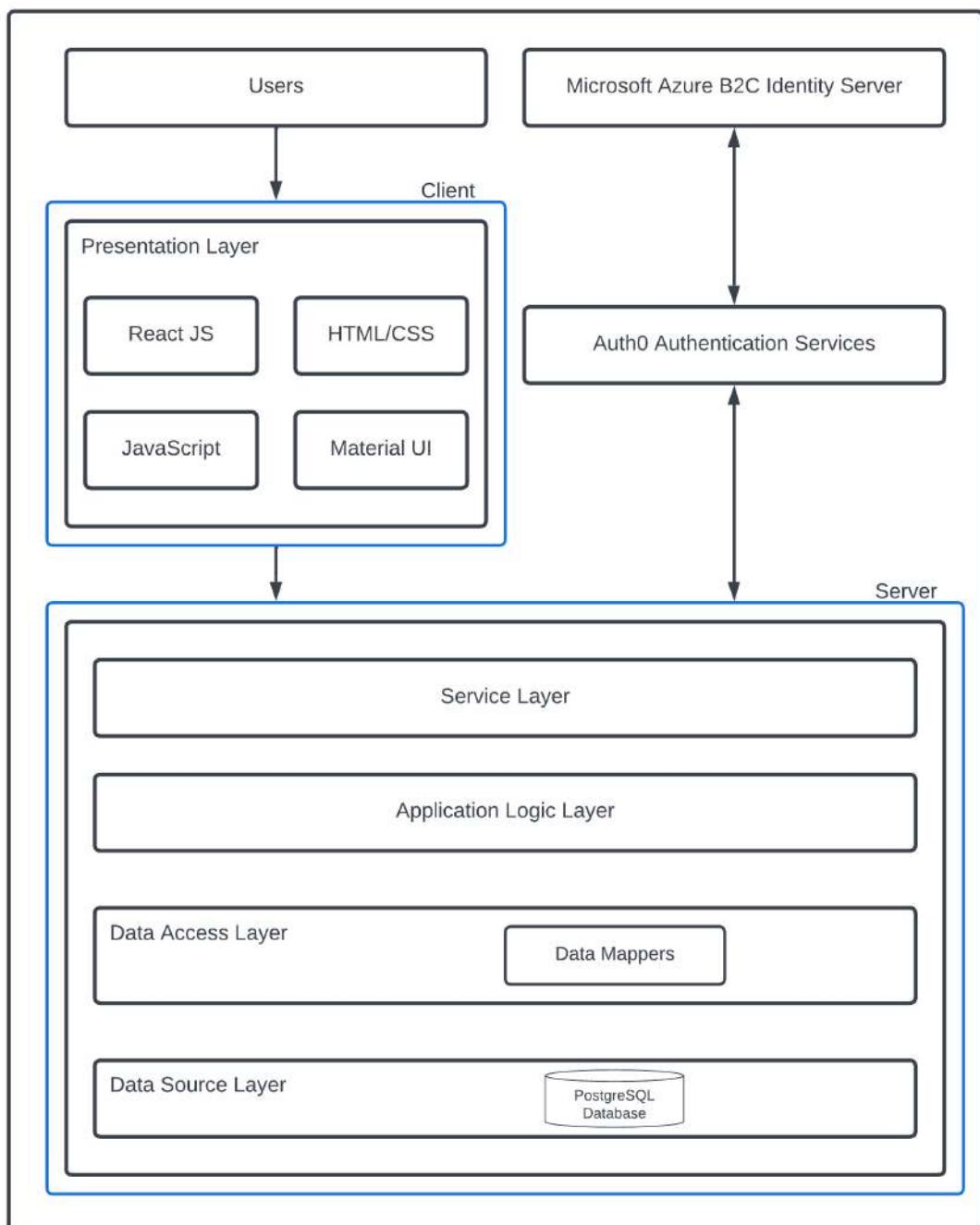


Figure 4.14: Logical layer

Software Design Pattern

The student talent development center web application will be developed using the MVC design pattern.

Model-View-Controller (MVC)

It is a design pattern is a software architecture pattern that separates the representation of information from the user's interaction. This pattern divides the application into 3 main components: model, view, and controller.

1. The model represents data and business logic of the application, It is responsible for managing the data, validating user input, and performing any other business-related tasks. This component resides in the server (the backend).
2. The view represents the user interface of the application. It is responsible for displaying the data to the user and providing a way for the user to interact with the application. This component resides in the client (the frontend).
3. The controller is the bridge between the model and the view. It receives user input, communicates with the model to perform any necessary actions, and then updates the view with the results. This component resides in the server (the backend).

The MVC design pattern helps to improve the separation of concerns in the application and makes it easier to maintain the different parts of the application independently.

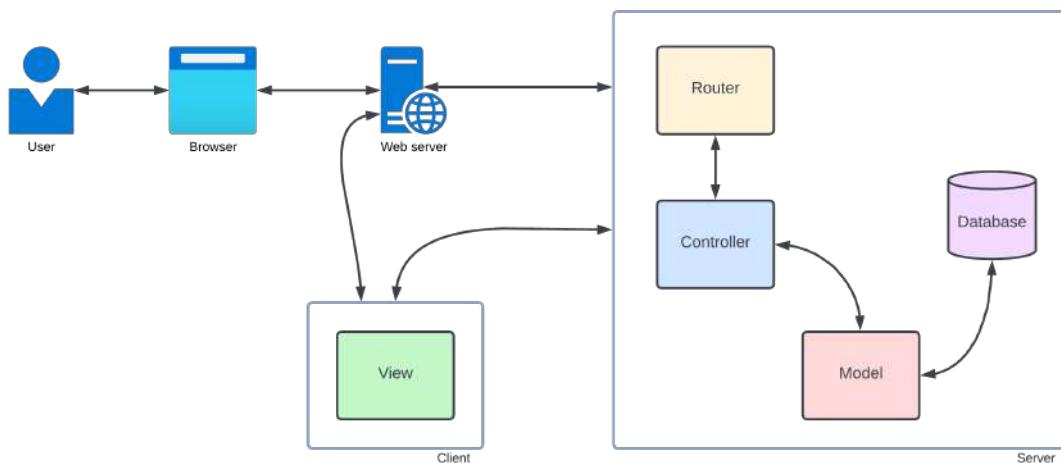


Figure 4.15: MVC pattern

Implementation Paradigms

The student talent development center web application will be developed using two different programming paradigms. The backend will be developed using Ruby on Rails framework by implementing the Object-Oriented Programming concepts. The frontend will be developed using React JS framework by implementing the Functional Programming concepts. Functional programming and object-oriented programming (OOP) are two programming paradigms, or approaches to writing software.

The evaluation of mathematical functions is the basis for computing in the functional programming paradigm. It is based on the idea of using functions to model computation, and it is characterized by the use of immutable data, higher-order functions, and a focus on pure functions. In functional programming, functions are first-class citizens, which means that they can be passed as arguments to other functions and returned as values from functions.

Object-oriented programming is a programming paradigm which is based on the concept of "objects," which represent data and the functions that operate on that data. In OOP, objects are created from templates called classes, and they can interact with each other through methods. OOP is characterized by the use of encapsulation, inheritance, and polymorphism.

Both functional programming and OOP have their own strengths and can be useful in different contexts. Therefore, both of them are chosen to be used in combination to one another.

4.3.2 Program Design

The following is the Program Design of the Student Talent Development Center Web Application. First, attention is given to the class modelling that consist of both dynamic classes and static classes. Then the relationship between classes is highlighted. After that, design pattern for each class is chosen. Lastly sequence models are illustrated to further explain the workflow of some of the complex areas of the application that have not been highlighted in the Use Case tables.

Class Model (Class Diagram)

The following class diagram contains both dynamic and the static classes. The Controller classes have a direct association with their corresponding classes (which are model classes). It has not been shown the diagram below for it will make the diagram very complex, however direct association exist between those controllers and model classes.

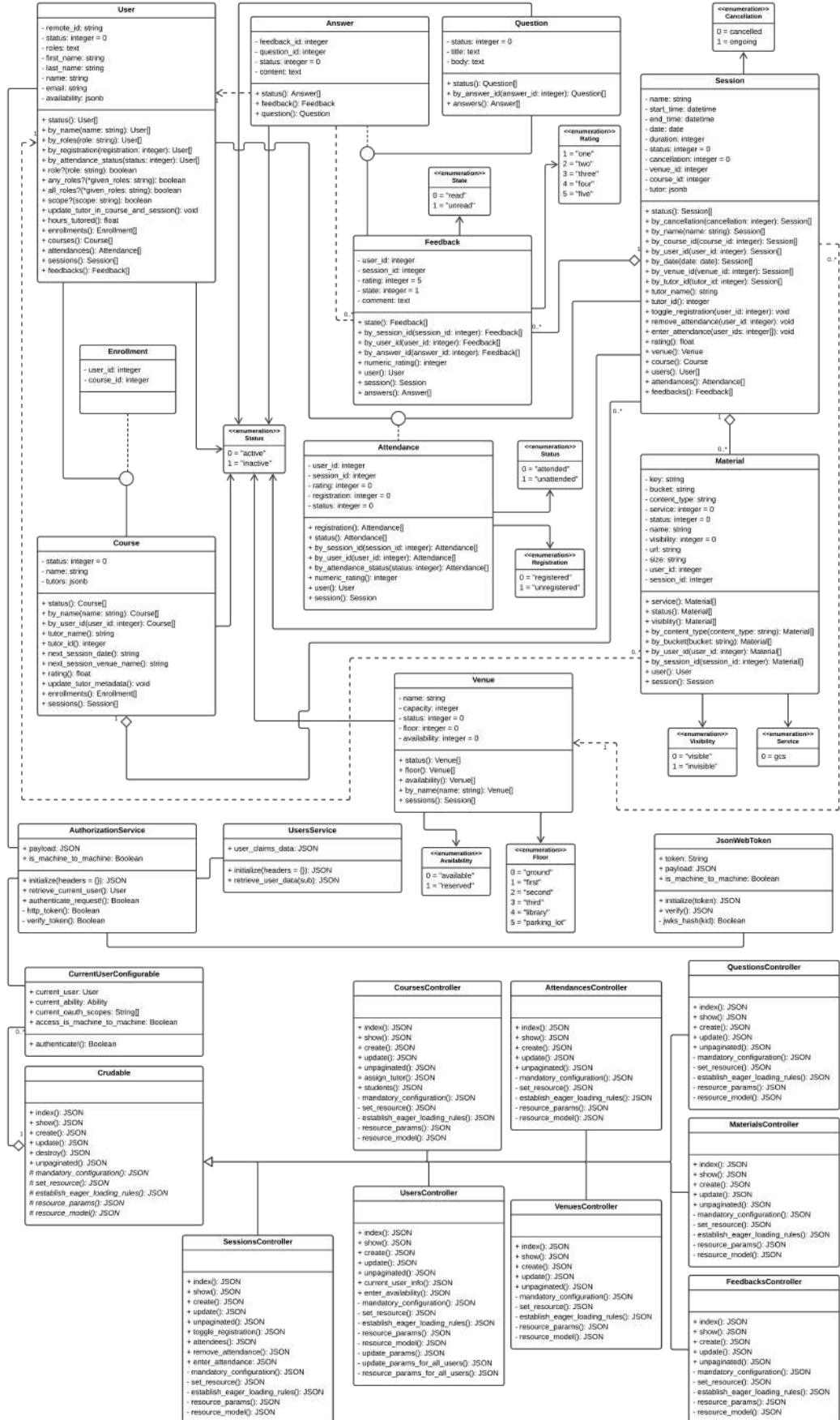


Figure 4.16: Class model – dynamic and static classes

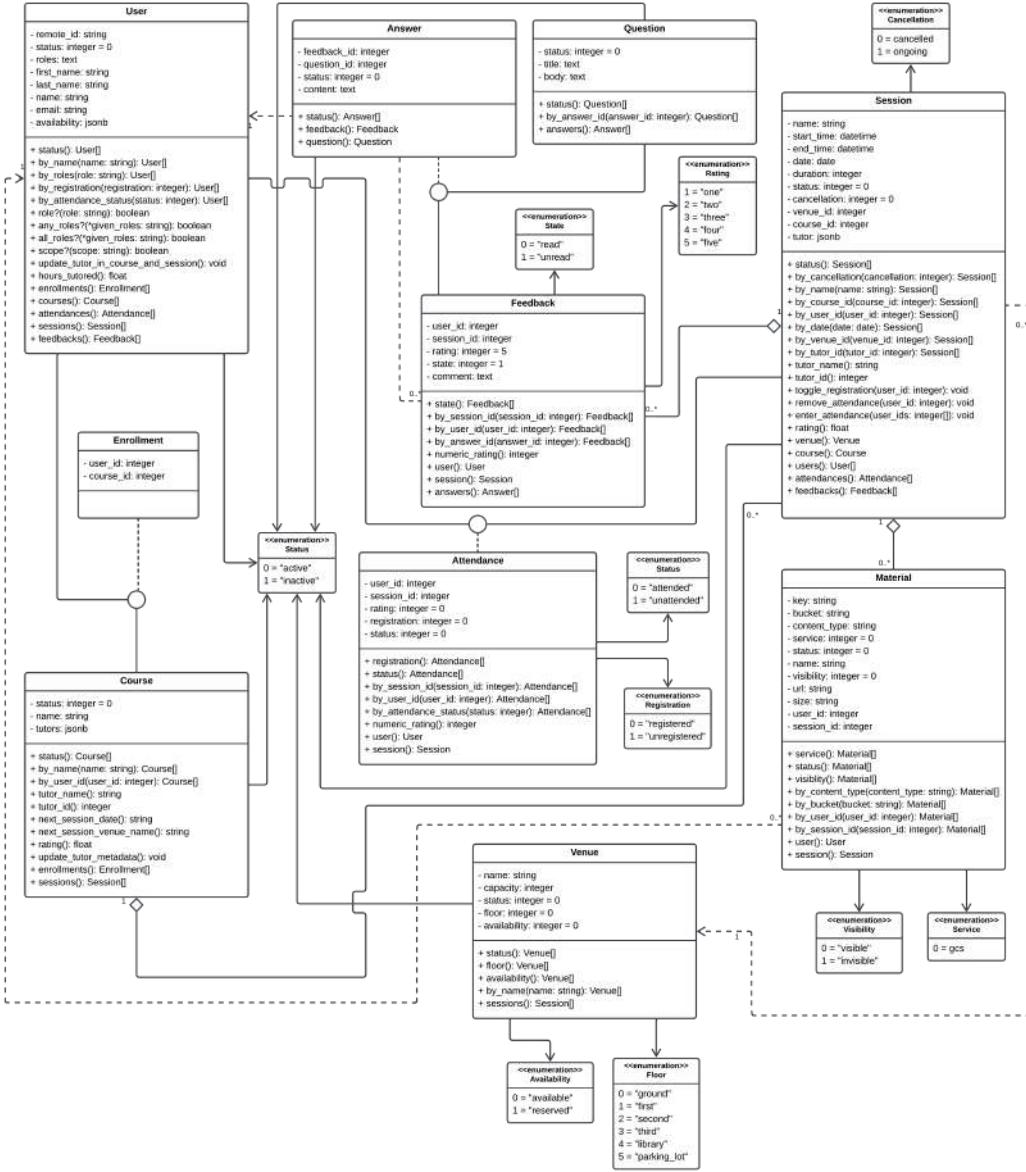


Figure 4.17: Class model – static classes

Class Design Pattern

The following is the chosen design pattern for each class in the class diagram. A design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Table 4.4: Class design patterns

Class Name	Design Pattern
User	Active Record
Course	Active Record
Enrollment	Structural Pattern
Feedback	Active Record
Attendance	Active Record
Answer	Active Record
Question	Active Record
Venue	Active Record
Material	Active Record
Session	Active Record
AuthorizationService	Service
JsonWebToken	Factory
CurrentUserConfigureable	Interactor
UsersService	Service
Crudable	Template Method
AttendancesController	Front Controller
UsersController	Front Controller
CoursesController	Front Controller
FeedbacksController	Front Controller
VenuesController	Front Controller
SessionsController	Front Controller
MaterialsController	Front Controller

Sequence Model (Sequence Diagram)

The followings are two sequence models (diagrams), first explaining how a current user in the application is set, and the second is the role of identity provider in the application in action. The rest does not require a sequence diagram for they have been clearly explained using the use case tabular form. The steps there with its reference to the user interface makes the process clear and easy to understand.

The first sequence diagram is the identity provider in action and shows how a request is authenticated by verifying the public keys and the access token received. The second sequence diagram is to show the process of setting the current user (login alike process).

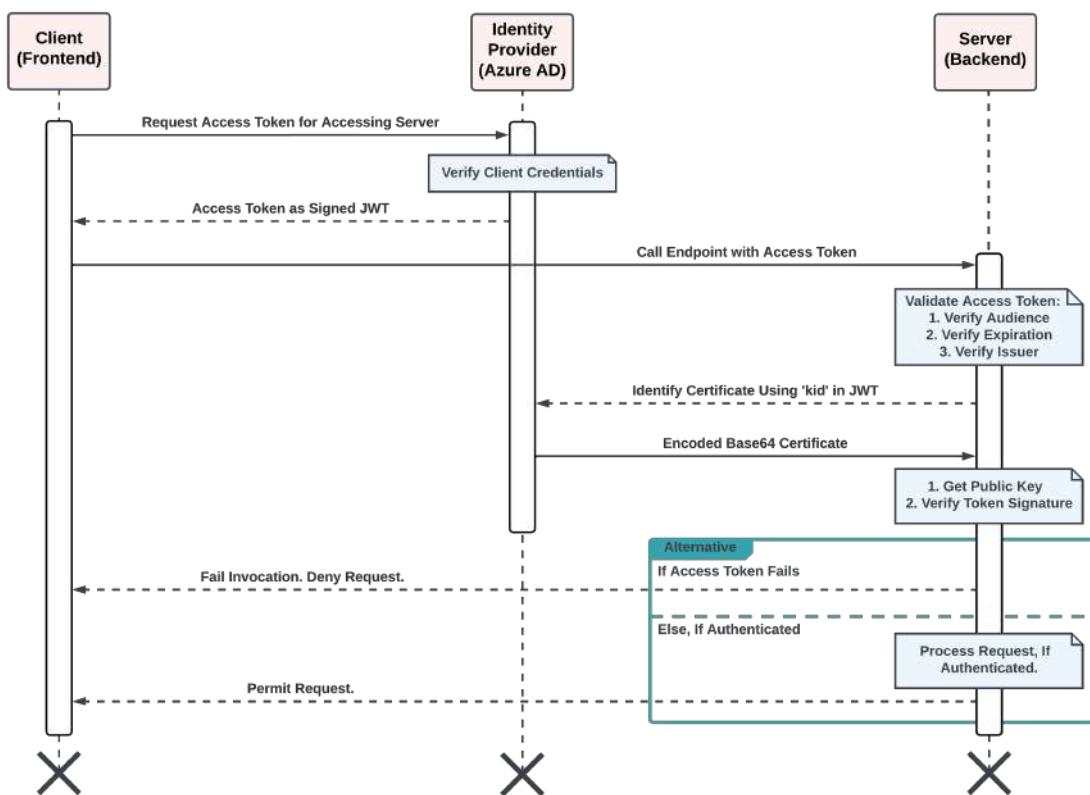


Figure 4.18: Sequence model – identity provider in action

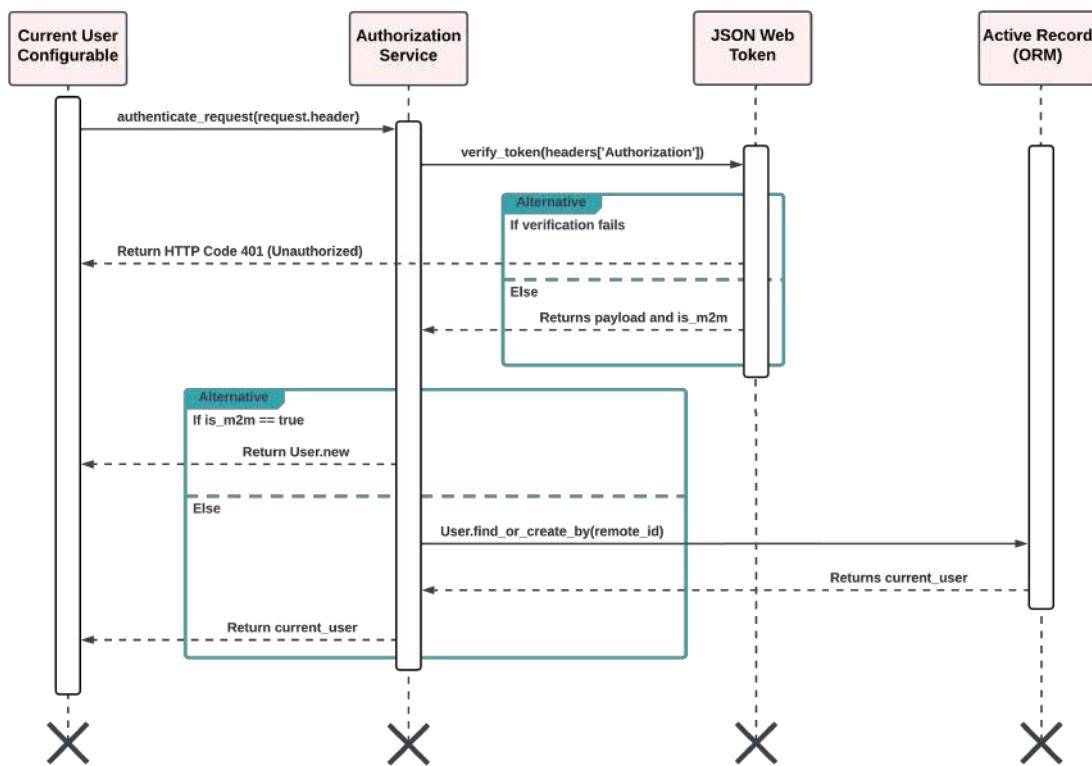


Figure 4.19: Sequence model – setting the current user

4.3.3 Interface Design

The following is the User Interface Design of the Student Talent Development Center Web Application. Though the application will end up having more specific user interfaces, below here the main pages of the application is illustrated along with landing page, and a form page that is used for creation, updating, or adding. The below user interfaces are created using FIGMA and there are few important assumptions that needs to be considered when viewing these user interfaces:

1. All the user interfaces are from the perspective of Admin. Except for “Profile” which is from the perspective of tutor.
2. Each user interface has an associated id that is used for referencing purposes.
3. When these user interfaces are viewed as student or tutor, they will be same except for the components that are not authorized for them to see. In that case, components are simply hidden away from the user without them noticing anything, this includes pages as well (e.g., Venues Page)
4. The last page (UI11) is the template that is going to be used when there is form like page to show, for example, creating, updating, submitting feedback, and many others which require a form to be submitted.

UI01. Landing Page (Login)

This page appears when user open the website for the first time. Users should login in order to see the application. Login is done through Microsoft using a UKH account. When user clicks login, they will be redirected to Microsoft Login page, where they will properly login and will be redirect back to the application's home page (UI02). The page below shows the landing page of the application.

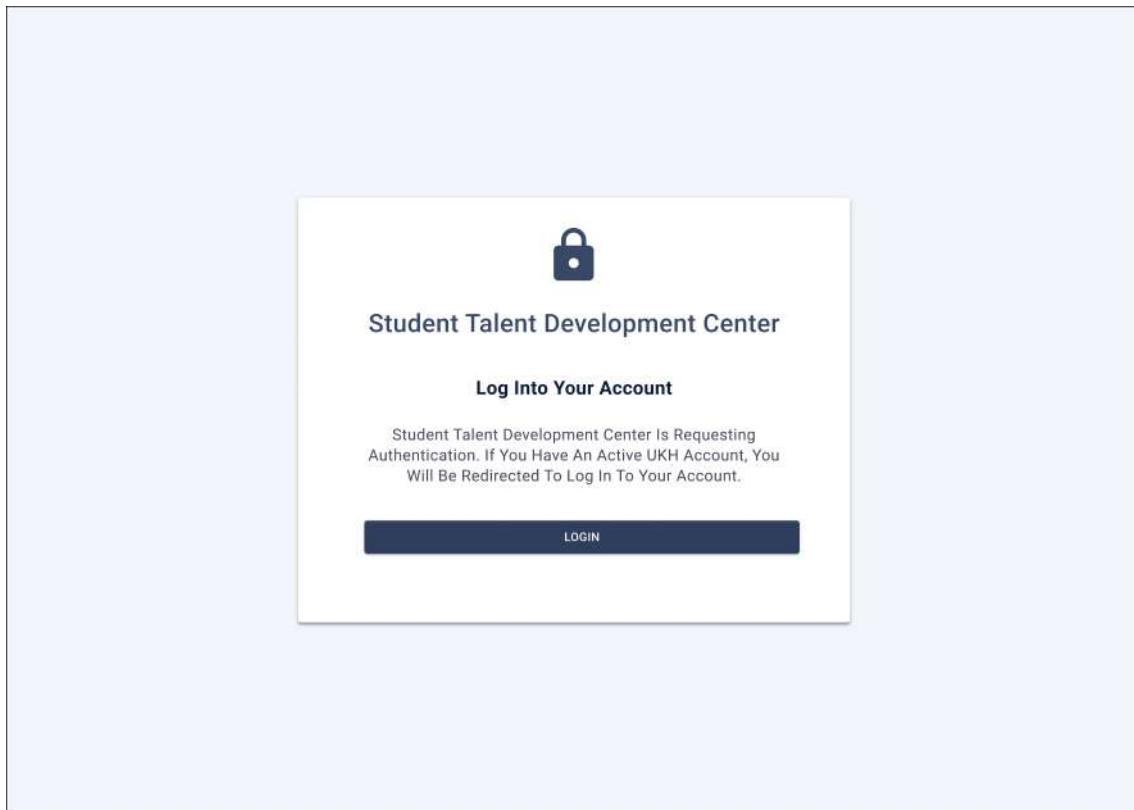


Figure 4.20: UI01. landing page (login)

UI02. Home Page

This page appears after user logs in, or when user clicks on “Home” button from the sidebar. This page shows a few important details, first the enrolled modules of the user. Second, it shows all the other modules that the user could enroll himself into. Users could search for modules by their name using the search bar. Users could also view any module in detail when they click on “View Module” button, this will redirect them to (UI09).

The screenshot displays the 'Student Talent Development Center' home page. On the left, a sidebar menu includes: Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main content area is divided into two sections: 'Enrolled Modules' and 'All Modules'.
Enrolled Modules: This section shows three modules. Each module card includes a thumbnail (a clock icon), the module name ('Next Session on 12th'), the author ('Joe Smith'), and a 5-star rating. Below each card is a 'View Module' button.

Module Name	Module Name	Module Name
Next Session on 12th Joe Smith ★★★★☆	Next Session on 12th Joe Smith ★★★★☆	Next Session on 12th Joe Smith ★★★★☆
View Module		

All Modules: This section displays a grid of 10 modules. Each module card follows the same structure: thumbnail, module name, author, and 5-star rating, with a 'View Module' button below. A search bar labeled 'Search...' is positioned above the grid.

| Module Name |
|--|--|--|--|--|
| Next Session on 12th
Joe Smith
★★★★☆ |
| View Module | | | | |
| Module Name |
| Next Session on 12th
Joe Smith
★★★★☆ |
| View Module | | | | |

At the bottom left, the footer reads: University Of Kurdistan Hewlêr, Student Talent Development Center, 1.0.0.

Figure 4.21: UI02. home page

UI03. Profile Page

This page appears when user clicks on “Profile” button from the sidebar. This page shows a few important details, first the user’s details shown in the table below, including the user’s email, first name, last name, roles, and status of their account. Second, it shows all the hours that the tutor has tutored (this part doesn’t show when user views profile). Third, the amount of money the tutor to be paid is show to the user. Lastly, the enter availability button which take user to a form page like UI11, where tutor can enter their availability. This button is not visible to students or admin.

The screenshot displays the 'Profile' page of the Student Talent Development Center. On the left, a sidebar lists navigation options: Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main content area is titled 'Profile' and features a user profile card for 'Joe Smith' (joe.smith@email.com). The card indicates 'Active' status and 'Tutor' role. To the right of the card are two summary boxes: one showing '\$115 Payment' and another showing '23 Hours Tutored'. Below the profile card is a table titled 'Personal Data' containing the following information:

Property	Information
First Name	Joe
Last Name	Smith
Email	joe.smith@email.com
Roles	Tutor
Status	Active

A blue button labeled '+ ENTER AVAILABILITY' is located at the bottom right of the data table. At the bottom of the page, there is a footer with the text 'University Of Kurdistan Hewlêr Student Talent Development Center' and a version number '1.0.0'.

Figure 4.22: UI03. profile page

UI04. Modules Page

This page appears when user clicks on “Modules” button from the sidebar. This page shows all the existing modules. It allows the user (admin) to edit or delete a module. Additionally, the assign tutor button allows admin to assign a module to tutor (a form alike page will appear when clicked). User can search for modules based on name, and user can filter modules, the fields for both is abstracted away, when implemented, there might be many of these filtering fields. It is important to note that this page is only visible to admin, so is the button on the sidebar.

The screenshot shows the 'Modules' page of the Student Talent Development Center. The left sidebar includes links for Home, Profile, Modules (which is selected), Sessions, Feedbacks, Venues, and Users. The main content area has a title 'Modules' and a 'Filters' section with a search input and dropdown for 'Attribute' (set to 'Property'). Below is a table with columns: Name, Tutor, Status, Rating, and Actions. The table lists 13 rows of modules, each with a 'Module Name' (e.g., 'Module Name'), 'Tutor' (e.g., 'Joe Smith'), 'Status' (e.g., 'Active' or 'Inactive'), 'Rating' (e.g., '4.6 / 5.0' or '3.1 / 5.0'), and 'Actions' (Edit and Delete buttons). At the bottom, it shows 'Rows per page: 10' and '1-5 of 19'.

Name	Tutor	Status	Rating	Actions
Module Name	Joe Smith	Active	4.6 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Active	4.6 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Active	4.6 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Active	4.6 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Active	4.6 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete
Module Name	Joe Smith	Inactive	3.1 / 5.0	Edit Delete

Figure 4.23: UI04. modules page

UI05. Sessions Page

This page appears when user clicks on “Sessions” button from the sidebar. This page shows all the upcoming sessions on top and then all of the existing sessions below that. It allows the user to view time and venue in short details. However, users have the ability to click on “View Session” button for further details (UI10). Users can also click on the “View Session History” button to view the history of sessions attended. Users can also search for sessions using the search bar provided.

The screenshot displays the "Student Talent Development Center" application interface. On the left is a sidebar with icons and links: Home, Profile, Modules, Sessions (selected), Feedbacks, Venues, and Users. The main content area has two sections: "Upcoming Sessions" and "All Sessions".

Upcoming Sessions: This section contains four cards, each representing an upcoming session. Each card includes a timestamp (Wednesday at 16:00), a participant name (Joe Smith), and a location (T13). Below each card is a "View Session" button.

Session Name	Session Name	Session Name	Session Name
Wednesday at 16:00 Joe Smith T13 View Session			

All Sessions: This section contains five rows of session cards. Each row has five cards, identical to the ones in the "Upcoming Sessions" section. A search bar labeled "Search..." is located above the second row of cards.

| Session Name |
|--|--|--|--|--|
| Wednesday at 16:00
Joe Smith
T13
View Session |
| Wednesday at 16:00
Joe Smith
T13
View Session |
| Wednesday at 16:00
Joe Smith
T13
View Session |
| Wednesday at 16:00
Joe Smith
T13
View Session |

Page Footer: The footer contains the text "University Of Kurdistan Hewlêr Student Talent Development Center" and the version "1.0.0".

Figure 4.24: UI05. sessions page

UI06. Feedbacks Page

This page appears when user clicks on “Feedbacks” button from the sidebar. This page shows all the received feedbacks. It allows the user (admin) to view feedback using the specified button. Additionally, admin can search for feedbacks and can filter feedbacks, the fields for both is abstracted away, when implemented, there might be many of these filtering fields. It is important to note that this page is only visible to admin, so is the button on the sidebar.

The screenshot shows the "Student Feedbacks" page within the "Student Talent Development Center". The left sidebar includes links for Home, Profile, Modules, Sessions, Feedbacks (which is the active tab), Venues, and Users. The main content area has a header "Student Feedbacks" and a "Filters" section with a search bar and dropdown menus for "Attribute" and "Property". Below this is a table listing eight entries, each representing a student with columns for Name, Email, Status (Active or Inactive), Roles (Student, Tutor, Admin), and Actions (View Feedback). The bottom of the page includes a footer with the text "University Of Kurdistan Hewlêr, Student Talent Development Center" and a version number "1.0.0".

Name	Email	Status	Roles	Actions
Joe Smith	test@gmail.com	Active	Student	<button>View Feedback</button>
Joe Smith	test@gmail.com	Inactive	Tutor	<button>View Feedback</button>
Joe Smith	test@gmail.com	Active	Student	<button>View Feedback</button>
Joe Smith	test@gmail.com	Inactive	Admin	<button>View Feedback</button>
Joe Smith	test@gmail.com	Active	Admin	<button>View Feedback</button>
Joe Smith	test@gmail.com	Inactive	Student	<button>View Feedback</button>
Joe Smith	test@gmail.com	Active	Student	<button>View Feedback</button>
Joe Smith	test@gmail.com	Inactive	Tutor	<button>View Feedback</button>

Figure 4.25: UI6. feedbacks page

UI07. Venues Page

This page appears when user clicks on “Venues” button from the sidebar. This page shows all the existing venues. It allows the user (admin) to edit or delete a venue. Additionally, the new button allows admin to add a new venue (a form alike page will appear when clicked). User (admin) can search for venues based on name, and user can filter venues, the fields for both is abstracted away, when implemented, there might be many of these filtering fields. It is important to note that this page is only visible to admin, so is the button on the sidebar.

The screenshot shows the 'Venues' page of a web application. The left sidebar includes links for Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main content area has a header 'Student Talent Development Center'. Below it, a 'Filters' section contains a search input ('Name, floor, etc...') and an attribute dropdown ('Property'). A '+ NEW' button and a settings icon are also present. The main table lists eight venues:

Name	Capacity	Status	Floor	Actions
G5	20	Active	Ground	<button>Edit</button> <button>Delete</button>
F11	30	Inactive	First	<button>Edit</button> <button>Delete</button>
S16	30	Active	Second	<button>Edit</button> <button>Delete</button>
T13	18	Inactive	Third	<button>Edit</button> <button>Delete</button>
Staff Cafeteria	40	Active	First	<button>Edit</button> <button>Delete</button>
L1	10	Inactive	Library	<button>Edit</button> <button>Delete</button>
P1	25	Active	Parking Lot	<button>Edit</button> <button>Delete</button>
Cafeteria	150	Inactive	First	<button>Edit</button> <button>Delete</button>

At the bottom, there are pagination controls and a message: 'Rows per page: 10' and '1-5 of 13'.

Figure 4.26: UI07. venues page

UI08. Users Page

This page appears when user clicks on “Users” button from the sidebar. This page shows all of the enrolled users. It allows the user (admin) to update a user’s information, or to add a user. Additionally, the “Onboard Tutor” button allows admin to onboard a user as a tutor (a form alike page will appear when clicked). Admin can search for users based on name, and can filter users, the fields for both is abstracted away, when implemented, there might be many of these filtering fields. It is important to note that this page is only visible to admin, so is the button on the sidebar.

The screenshot shows the 'Users Management' page of the Student Talent Development Center. The left sidebar includes links for Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users (which is selected). The main area has a header 'Student Talent Development Center' and a 'Filters' section with a search input ('Name, email, etc...'), an attribute dropdown ('Property'), and buttons for '+ ONBOARD TUTOR', '+ ADD', and settings. Below is a table of users:

Name	Email	Status	Roles	Actions
Joe Smith	test@gmail.com	Active	Student	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Inactive	Tutor	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Active	Student	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Inactive	Admin	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Active	Admin	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Inactive	Student	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Active	Student	<button>Edit</button> <button>Delete</button>
Joe Smith	test@gmail.com	Inactive	Tutor	<button>Edit</button> <button>Delete</button>

At the bottom, it says 'Rows per page: 10' and '1-5 of 13' with navigation arrows.

Figure 4.27: UI08. users page

UI09. View Module Page

This page appears when user clicks on “View Module” button from the modules page. This page shows all of the information of a module, and show all of its sessions. It also allows the users view its properties in a tabular format when “properties” is clicked. Additionally, the “Assign Tutor” button allows admin to assign a tutor for a module (a form alike page will appear when clicked). Admin can search for sessions using the search bar, and can view the sessions. It is important to note that the button on the top right “Assign Tutor” is only available to admin.

The screenshot displays the 'View Module Page' interface. At the top, there's a header with the text 'Student Talent Development Center'. On the left, a sidebar menu includes links for Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main content area is titled 'Module Name' and contains tabs for 'Sessions' and 'Properties'. A prominent 'Assign Tutor' button is located in the top right corner. Below the tabs, a search bar with a magnifying glass icon and the word 'Search' is present. The main content area features a grid of session cards. Each card displays session details such as 'Session Name', 'Wednesday at 16:00', 'Joe Smith', and 'T13'. Below each card is a blue 'View Session' button. The grid is organized into three rows and five columns. In the bottom left corner of the page, there's a footer with the text 'University Of Kurdistan Hewlêr Student Talent Development Center' and '1.0.0'.

Figure 4.28: UI09. view module page

UI10. View Session Page

This page appears when user clicks on “View Session” button from the modules page and sessions page. This page shows all of the information of a session. Additionally, users have access to a few tabs of the session including materials, attendance, provide feedback, assign time and venue, and upload material with addition to the cancel session button on top right corner. These are all different actions to take when this page is viewed. Note that not all of these are available to all the users. Cancel Session is allowed for Tutor, Assign Time and Venue is only allowed for admin, Provide Feedback only to student, and so on.

The screenshot displays the 'Student Talent Development Center' interface. On the left is a sidebar with icons for Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main header bar includes 'View Registered Students' and 'Cancel Session' buttons, along with session details: 'Wednesday at 16:00', 'Joe Smith', and 'T13'. Below the header, the title 'Module Name - Session #21' is centered. A horizontal menu bar contains links for Materials, Attendance, Provide Feedback, Assign Time and Venue, Upload Material, and Register. The 'Materials' section is titled 'Material' and lists eight PDF documents, each with a delete ('X') and edit ('C') icon. The footer contains the text 'University Of Kurdistan Hewlêr Student Talent Development Center' and a version '1.0.0'.

Figure 4.29: UI10. view session page

UI11. Form Page

This page appears after user is faced with a form alike page. This includes creation, adding, updating, editing, providing feedback, entering availability, assignment features, and so on. The properties of this page vary depending on the form fields; however, the representation below should give a good overall structure and look of these forms. When a required field is left empty, the field outline turns red and the text “required” appear under that field.

The screenshot shows a web-based application interface for the "Student Talent Development Center". On the left is a vertical navigation sidebar with icons and links: Home, Profile, Modules, Sessions, Feedbacks, Venues, and Users. The main content area has a title "New/Update/Add Entity". It contains several input fields for attributes, some of which are highlighted with red borders and labeled "required". A dropdown menu is open on the right side, showing options like "Menu Item". At the bottom of the page, there is footer text: "University Of Kurdistan Hewlêr", "Student Talent Development Center", and a version number "1.0.0".

Figure 4.30: UI11. form page

4.3.4 Database Design

The following is the Database Design of the Student Talent Development Center Web Application. First, the strategy for mapping the object to relational tables is discussed. Secondly, attention is given to the logical modelling where the persistent entities are defined, then the relationship between entities is highlighted. Lastly, the physical modelling will be illustrated where the data types of the attributes are set.

Mapping Objects to Relational Tables

The Student Talent Development Center Web Application is constructed being developed using Object oriented principles and utilizes a relational database management system (RDBMS) to store data persistently. The connecting of objects and tables is going to be handled by an "object-relational mapping layer" (ORM) that will be a part of STDC. The primary goal of this layer would be to simplify and speed up the mapping procedure while also providing an abstraction that would insulate users from the implementation specifics of the mapping. One aspect of this is the relationship and data type mapping. This mapping procedure is, to a certain degree, simple. Object-oriented class inheritance, also known as generalization, stands out as a relationship type that needs extra care.

Property Mapping

It is decided that each persistent attribute of an object will be mapped to a column in the database tables. This will ensure that object properties are mapped correctly inside the system. To put it more simply, a mapping of a single column to a single attribute. The attributes, also known as properties, are considered to be persistent when they maintain their presence after the execution process has been completed. Additionally, metadata will be persisted and kept in the database as they are in the object.

Relationship Mapping

In spite of the fact that the student talent development center does not yet have persistent class inheritance at this stage, it is essential to choose a mapping technique for inheritance classes before proceeding with any extension. It is possible to map inherited classes into their tables in one of three different ways: either by mapping

the hierarchy of classes to a single table, mapping each concrete class to its own table, or mapping each class to its own table. The latter method, in which each table is mapped to one of its own tables, is the one that will be used by STDC. Conceptually, this is the mapping that comes the closest to inheritance. In addition to understanding mapping of properties and inheritance, it is required to have a solid grasp of the relationship mapping of various kinds of associations. Association, aggregation, and composition are the three different kinds of object relationships that need to be mapped. Because of how they are mapped in this application, each and every one of them will be handled in the exact same manner. When it comes to mapping, STDC is interested in two different sorts of object relationships. These categories are: The first category is one that is founded on the concept of multiplicity, and it consists of three different sorts. Relationships that are one-to-one, those that are one-to-many, and those that are many-to-many are all possible. The second category is one that is focused on directionality, and within this category there are two different kinds of relationships: uni-directional and bi-directional relationships.

In conclusion, classes are mapped one to one with tables based on their multiplicity and directions, objects are mapped one to one if they are persistent after execution.

Database Logical Model

The below model is the Entity Attribute Relationship Diagram which is derived from the class diagram (in Program Design). Each entity is a persistent entity in the database along with their attributes. Note that enumeration classes are not referred to here in the EARD for the sake of simplicity of the model due to its repetitive nature. The relationship between each entity is further explained after the below figure.

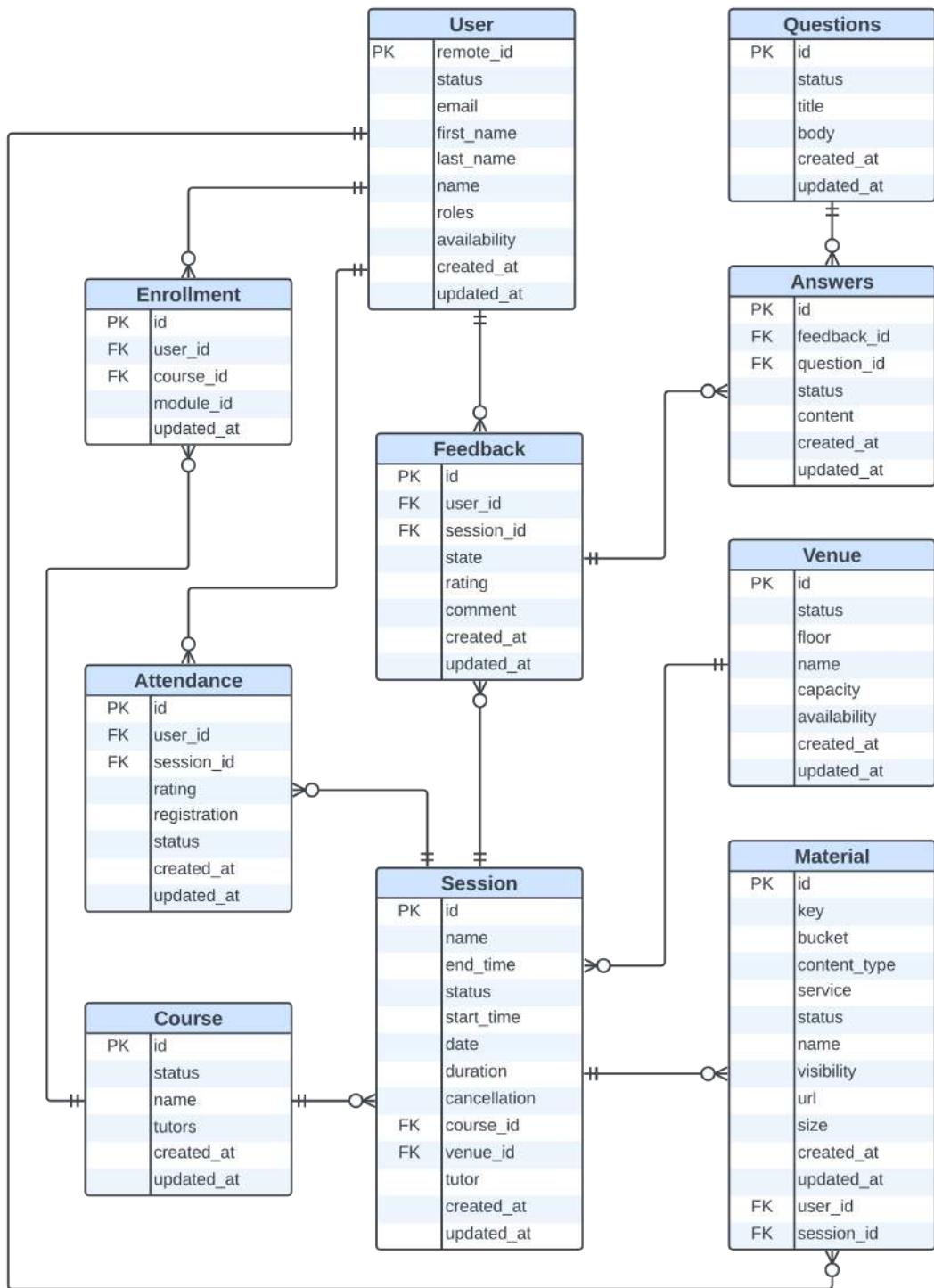


Figure 4.31: Database – logical entity attribute relationship diagram

1. User – Feedback

- Each user can provide zero or many feedbacks.
- Each feedback submitted belongs to one user and one only.

Each feedback received has a unique id. User cannot provide multiple

feedbacks to the same session.

2. User – Material

- a. Each user (tutor) can upload zero or many materials.
- b. Each material uploaded belongs to one user and one only.

Each material received has a unique id. User can upload multiple materials to the same session each with different id.

3. User – Course

- a. Each user (tutor) can tutor many courses.
- b. Each course is tutored by many users (tutors) considering the fact that in the upcoming years, tutor will change for a module. History of tutors of a course should be trackable for future extendibility.

The many to many relationships between these is resolved by having the third table Enrollment. Each entry here will have a unique id.

4. User – Session

- a. Each user (student) can attend zero or many sessions.
- b. Each session can have zero or many users (attendees).

The many to many relationships between these is resolved by having the third table Attendance. Each entry here will have a unique id.

5. Feedback – Question

- a. Each feedback can have zero or many questions.
- b. Each question can have zero or many feedbacks.

The many to many relationships between these is resolved by having the third table Answer. Each entry here will have a unique id.

6. Session -- Module

- a. Each session belongs to one module and one only.

- b. Each module has zero or many sessions.

7. Session -- Venue

- a. Each session has one venue and one only.
- b. Each venue has zero or many sessions.

8. Session -- Feedback

- a. Each feedback belongs to one and only one session.
- b. Each session can have zero or many feedbacks.

9. Session -- Material

- a. Each material belongs to one and only one session.
- b. Each session can have zero or many materials uploaded to it.

Normalization

It is worth mentioning that the database is already in its third normal form. It has been normalized.

Database Physical Model

The physical modelling will be illustrated where the data type of the attributes is set. The data types are based on the chosen Database Management System. The Student Talent Development Center Web Application will make use of PostgreSQL database management system as its main database vendor. PostgreSQL (also known as Postgres) is free and open-source object-relational database (ORDBMS) with features like table inheritance and function overloading. PostgreSQL is most optimized and native to the chosen backend stack ruby on rails. Therefore, it makes the most suitable DBMS for the system. Here is the EARD extended with its data types based on PostgreSQL.

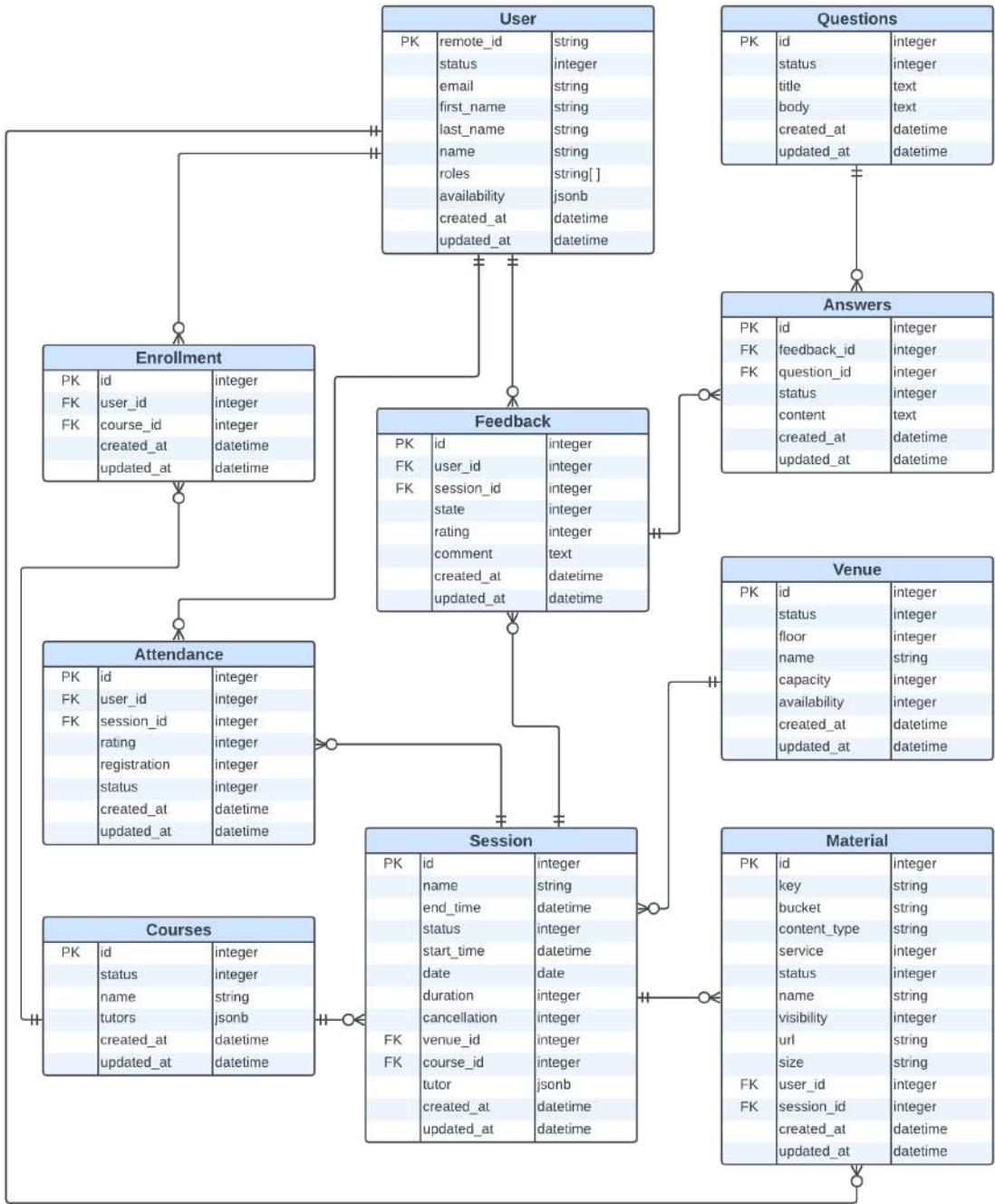


Figure 4.32: Database – physical entity attribute relationship diagram

Chapter 5 Implementation and Testing

This chapter delves into the Implementation, Testing, and Deployment phases of the software development process. Each of these phases plays a crucial role in bringing the project from conceptualization to realization. In this chapter, we will explore these phases in detail, examining the key activities and considerations involved in each.

The Implementation phase focuses on translating the design specifications into a functional software system. This involves writing code, configuring databases, integrating modules, and implementing various components that collectively bring the envisioned system to life. We will delve into the specific aspects of backend and frontend implementation, discussing the technologies used, validation mechanisms, controllers, models, database management, security practices, and more.

Following the implementation, the Testing phase ensures that the software meets the expected standards of quality, functionality, and reliability. We will explore the different types of tests conducted, such as unit tests, integration tests, and automated tests, to validate the system's behavior, uncover defects, and ensure smooth performance across various scenarios.

Once the software has successfully passed the testing phase, it proceeds to the Deployment phase, where it is made available to end-users. This phase involves setting up the necessary infrastructure, configuring servers, deploying the application, and addressing any compatibility or performance issues that may arise during the deployment process. We will examine the steps involved in deploying both the backend and frontend components, ensuring a smooth and seamless transition to the production environment.

5.1 Implementation

The details of how the Student Talent Development Center (STDC) project will actually be carried out are presented in the following section. The backend of the implementation process (also known as the API) was separated from the frontend of the process (also known as the client). The development strategy that was utilized placed a primary emphasis on the creation of a smart backend as well as a dummy frontend. According to the Architecture Design, the application's data logic and business logic are stored in the backend, whereas the frontend is primarily responsible for handling the presentation layer. The frontend also have some of the logic layer. By taking this approach, the backend will be transformed into the centralized hub for all of the business logic, which will ensure security and separation from the client side. The primary responsibility of the frontend, meanwhile, is to effectively display the data that has been retrieved from the backend server.

The implementation phase of the STDC project was of the utmost importance because it aimed to deliver a functional Minimum Viable Product (MVP) that was capable of fulfilling the documented requirements of the project's stakeholders. These requirements were identified during the Analysis phase of the project. The preliminary version of the STDC manages to incorporate all of the essential capabilities without any problems. Due to the limited amount of time available, certain features that were not absolutely necessary but were thought to be advantageous, such as keeping a record of attendees who were removed from the attendance list in a trash can, were not implemented. As a direct result of this, the overall quality of the user experience may not be as good as it could be because some of these "nice to have" features were not satisfied. Nevertheless, the primary objective of this phase was accomplished by automating the manual workflows of the center and incorporating the essential requirements outlined during the analysis.

The objectives and goals that were established for this phase of implementation were successfully accomplished, which resulted in the completion of both the backend and the frontend of the project. In the following pages, a comprehensive review of the specific implementation details for each component, including the technologies used

and the necessary setup to fulfill all of the requirements, will be provided.

It is essential to take into consideration the possibility that some portions of the application do not adhere to the coding practices that are considered to be the cleanest or the most optimal. Instead, the focus was placed on selecting the approaches that were the most appropriate and efficient in order to achieve the objectives that were desired. As a result of the limited amount of time available, the code cleanup and revision were not possible. In spite of this, it is important to point out that the application is completely operational and functional in every way.

5.1.1 Backend Implementation (API)

The following section shows how the implementation for the Server (the backend) has been done for the Student Talent Development Center Web Application.

Technologies Used

The Student Talent Development Center (STDC) application makes use of a number of different technologies in its backend to ensure that its operations are carried out in an effective and safe manner.

The API has been developed using Ruby on Rails as its foundational framework. Because of its widely used and opinionated character, it offers a robust and convention-driven environment for software development. The Database makes use of PostgreSQL, a relational database that is both reliable and plentiful in features. The database management tasks it handles are made easier by its seamless integration with Ruby on Rails.

Redis is a high-performance key-value store that is well-known for its speed and versatility. The application makes use of Redis in order to handle background jobs and caching. Sidekiq is a widely used framework for performing background processing, and it manages and carries out these jobs effectively. Sidekiq is also using the Redis database for its needs.

JSON Web Tokens (JWT), a secure standard that is widely used for web applications, are what the API makes use of for user authentication. OpenID Connect is yet another authentication standard that is widely recognized, OpenID Connet makes use of OAuth2.0 Standards. Both are used for user authentication and enrollment which is useful since the API does not store those user credentials.

Cancancan is a popular gem that integrates seamlessly with Ruby on Rails and provides capabilities for straightforward and granular user authorization. It makes the authorization of users much simpler. For the purpose of generating comprehensive API documentation based on integration tests, Rswag, which is built on OpenAPI

and Swagger UI, is utilized.

The backend makes use of a wide variety of tools and libraries in order to make testing procedures easier. In the context of testing, in the STDC application, the testing framework of choice is RSPEC, which has gained popularity for both unit tests and integration tests. In addition, Faker is responsible for the generation of fake data, while FactoryBot is the tool that is utilized to effectively create test data. Shoulda-matchers simplify the testing process by making it possible to create test cases that are concise and expressive. Database Cleaner makes certain that the database is cleaned up after each test, thereby promoting the independence and reliability of the tests.

The quality of the code is kept high by using a tool called Rubocop, which is a code linter. This tool ensures that coding standards are followed. In order to achieve seamless configuration management, dotenv is used to load environment variables. The API's health is monitored by Rails healthcheck, and Bootsnap speeds up the API's boot time so that it can be tested more quickly.

The backend makes use of Rack-cors to handle cross-origin resource sharing (CORS), which allows requests from other domains to be processed in accordance with the rules that have been configured. Rspec-sonarqube-formatter produces an exhaustive report for Sonarqube, whereas Simplecov monitors the amount of code that is tested throughout the development process.

Rails_12factor is what makes it possible to deploy the API to Heroku in production environments. Additionally, the Google-cloud-storage makes it easier to store files in Google Cloud Storage. Connection_pool is responsible for the efficient management of database connections, which is essential for achieving peak performance. Puma operates as the web server and responds to requests that are sent its way.

Other noteworthy technologies used include Enum_attributes_validation for validating enumerable attributes, Sidekiq-cron for scheduling background jobs, API-pagination and Kaminari for paginating API responses, and JSONAPI-serializer for serializing API responses in the JSON API format. These technologies validate enumerable

attributes, schedule background jobs, and paginate API responses, respectively.

In addition, the user authentication and user management functions are handled by an external service known as Auth0. This service acts as a secure middleware in the connection between the API and the Microsoft Azure Active Directory Service.

The implementation of the STDC application's backend is made more robust, secure, and efficient thanks to the utilization of these technologies, which also support the application's core functionalities and provide a seamless experience.

Authentication

In this section, we will discuss the security of the system, the tokens, and the OAuth 2.0. The aim of this section is to provide a clear understanding of the authentication process is handled in the system.

The Student Talent Development Center (STDC) makes use of OAuth 2.0 and OpenID Connect to handle its authentication. OAuth 2.0 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and Google. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications. This specification and its extensions are being developed within the Internet Engineering Task Force (IETF) which is a standardization organization.

The Student Talent Development Center (STDC) does not require a signing up process, since only UKH students will be using the application, it was decided that all users should be able to login using their UKH email (which is provided by Microsoft). The STDC web application has made use of OAuth 2.0 to authenticate users, and OpenID Connect to get the user information from Microsoft through Auth0 (which is a service that provides authentication and authorization as a service).

Before explaining how the authentication process is, let's first discuss how OAuth 2.0 and OpenID works briefly then we will discuss how the authentication process is handled in the STDC web application.

Earlier we have mentioned that OAuth is an authorization framework, which means that it is used to grant access to resources in one application to another application. For example, if you want to login to a website using your Facebook account, the website will ask you to grant it access to your Facebook account, and if you accept, the website will be able to access your Facebook account. This is how OAuth works, it allows you to grant access to your information to another application without giving it your password. OAuth 2.0 is the second version of OAuth, and it is the most widely

used authorization framework.

There are many different flows in OAuth 2.0, and each flow is used for a different purpose. The flow that is used in the STDC web application is the Authorization Code Flow. This flow is used when the application needs to access resources on behalf of the user, and it requires the user to login to the application. The flow works as follows:

- The user clicks on the login button on the STDC web application.
- The STDC web application redirects the user to Auth0.
- Auth0 redirects the user to Microsoft login page.
- The user enters his/her email and password.
- Microsoft redirects the user to Auth0.
- Auth0 redirects the user to the STDC web application with an authorization code.
- The STDC web application sends the authorization code to Auth0.
- Auth0 sends a request to Microsoft to exchange the authorization code for an access token.
- Microsoft sends the access token to Auth0.
- Auth0 sends the access token to the STDC web application.
- The STDC web application uses the access token to get the user information from Microsoft.
- The STDC web application creates a session for the user.
- The user is logged in.
- The user can now access the resources that he/she is authorized to access.

The below sequence diagram illustrates the above mentioned steps.

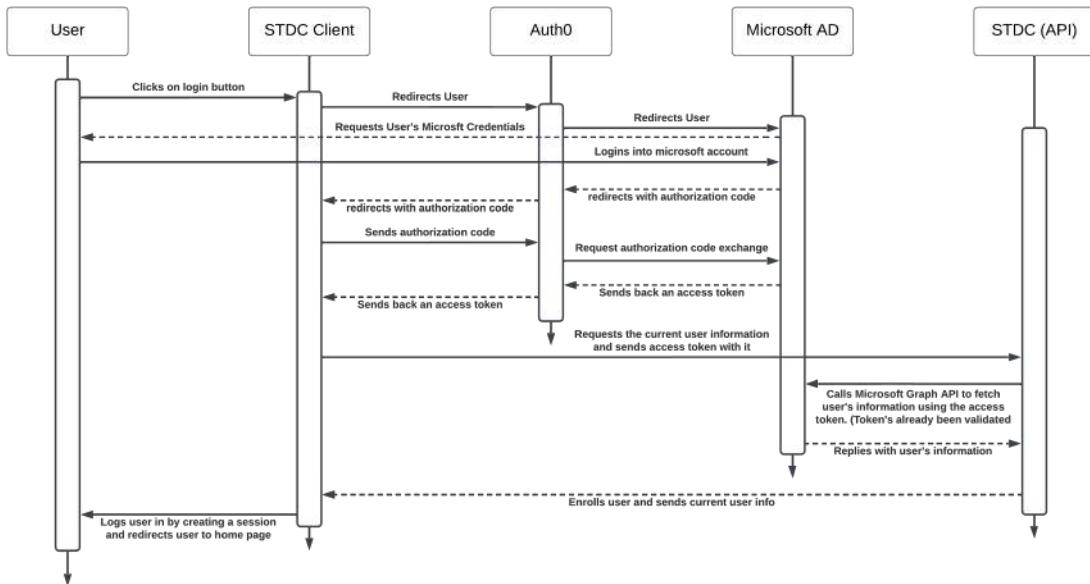


Figure 5.1: OAuth2.0 Authorization Code Flow - STDC's Authentication Process

Please note that the OpenID Connect is an authentication layer on top of OAuth 2.0. It allows the user to login to the application using an identity provider (such as Microsoft, Google, Facebook, etc.). OpenID Connect is used in the STDC web application to get the user information from Microsoft.

Now that we have discussed how OAuth 2.0 and OpenID Connect works, let's discuss how the authentication process is handled in the STDC web application's backend because the above flow is an abstraction of how the general authentication happens in both client and server. The authentication process of the server side (backend) of STDC works as follows:

- The API is a Restful API, which means that it is stateless, and it does not store any information about the user. This means that the API does not know if the user is logged in or not, therefore each request is considered as a new request. This requires the client to send an access token with each request to the API.
- The client sends a request to the API with an access token. Now the API needs to verify the access token to make sure that it is valid and not expired. The STDC will allow the client access the resources only if the user is authenticated. Below here we will dive into some of the classes and modules responsible for the authentication process.

- AuthorizationService
- UserService
- JsonWebToken
- GuardController
- CurrentUserConfigurable
- ApplicationController

GuardController



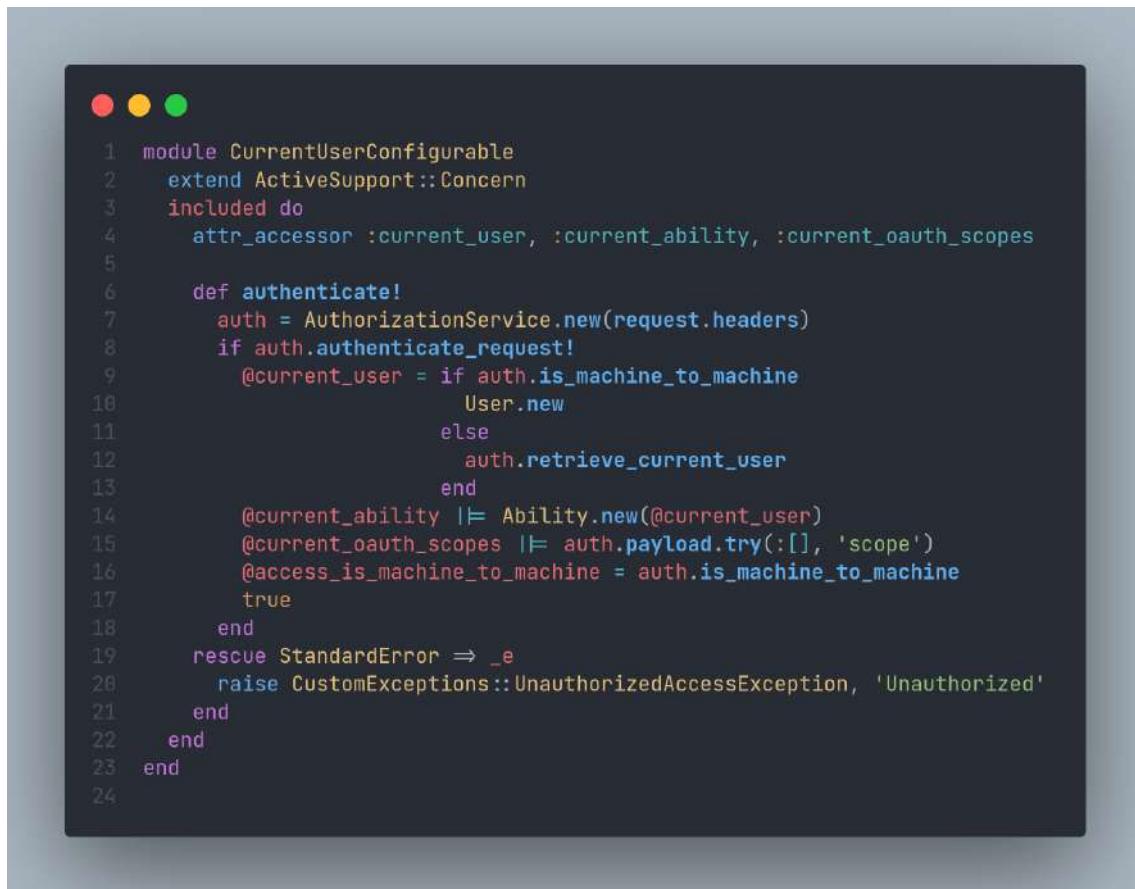
The screenshot shows a terminal window with a dark background. At the top left, there are three small colored circles: red, yellow, and green. Below them, the code for the GuardController is displayed:

```
1 class GuardController < ApplicationController
2 prepend_before_action :authenticate!
3 end
4
```

Figure 5.2: Authentication - GuardController

The GuardController is the base for every controller in the STDC application. The ApplicationController class which is a class defined by the Ruby on Rails framework is inherited by the GuardController. The GuardController calls a method called authenticate! before any action in any controller. The authenticate! method is defined in the CurrentUserConfigurable module which is included in the ApplicationController.

CurrentUserConfigurable



```
1 module CurrentUserConfigurable
2   extend ActiveSupport::Concern
3   included do
4     attr_accessor :current_user, :current_ability, :current_oauth_scopes
5
6     def authenticate!
7       auth = AuthorizationService.new(request.headers)
8       if auth.authenticate_request!
9         @current_user = if auth.is_machine_to_machine
10           User.new
11         else
12           auth.retrieve_current_user
13         end
14         @current_ability ||= Ability.new(@current_user)
15         @current_oauth_scopes ||= auth.payload.try([], 'scope')
16         @access_is_machine_to_machine = auth.is_machine_to_machine
17         true
18       end
19     rescue StandardError => _e
20       raise CustomExceptions::UnauthorizedAccessException, 'Unauthorized'
21     end
22   end
23 end
24
```

Figure 5.3: Authentication - CurrentUserConfigurable

The CurrentUserConfigurable is a module that is included in the ApplicationController. The authenticate! method is responsible for authenticating the user. It works as follows:

- The authenticate! method calls the authenticate_request! method in the AuthorizationService class.
- If the authenticate_request! method returns true, then the authenticate! method will set the current_user, current_ability, and current_oauth_scopes instance variables.
- If the authenticate_request! method returns false, then the authenticate! method will raise an exception.

AuthorizationService



```
1 class AuthorizationService
2   attr_accessor :payload, :is_machine_to_machine
3
4   def initialize(headers = {})
5     @headers = headers
6     @current_user = nil
7     @payload = {}
8   end
9
10  def retrieve_current_user
11    @current_user
12  end
13
14  def authenticate_request!
15    return unless verify_token
16    # Set Instance Variables from JsonWebToken
17    @is_machine_to_machine = @json_web_token.is_machine_to_machine
18    @payload = @json_web_token.payload
19
20    # No need to go further as M2M Tokens are not associated with users
21    return true if @json_web_token.is_machine_to_machine
22
23    # If found, set the current user
24    @current_user = if User.find_by(remote_id: @json_web_token.payload.try([], 'sub'))
25      User.find_by(remote_id: @json_web_token.payload.try([], 'sub'))
26    else
27      retrieved_user = UsersService.new(@headers).retrieve_user_data(@json_web_token.payload.try([], 'sub'))
28      User.create!(remote_id: retrieved_user['sub'],
29                  roles: ['viewer'],
30                  name: retrieved_user['name'],
31                  email: retrieved_user['email'],
32                  first_name: retrieved_user['given_name'],
33                  last_name: retrieved_user['family_name'])
34    end
35  end
36
37  # return true only if user is active for authentication]
38  return true if @current_user
39
40  # Always return false otherwise
41  raise StandardError
42 end
43
44 private
45
46  def http_token
47    @headers['Authorization'].split.last if @headers['Authorization'].present?
48  end
49
50  def verify_token
51    @json_web_token ||= JsonWebToken.new(http_token)
52    @json_web_token.verify
53  end
54 end
55
```

Figure 5.4: Authentication - AuthorizationService

The AuthorizationService class is responsible for authenticating the user. It works as follows:

- The AuthorizationService class has a method called `authenticate_request!` which is called by the `authenticate!` method in the `CurrentUserConfigurable` module.
- The `authenticate_request!` first calls the `verify_token` method to verify the access token.
- If the `verify_token` method returns true, then the `authenticate_request!` method will set the `payload` and `is_machine_to_machine` instance variables.

- If the `verify_token` method returns false, then the `authenticate_request!` method will raise an exception.
- If the `is_machine_to_machine` instance variable is true, then the `authenticate_request!` method will return true.
- If the `is_machine_to_machine` instance variable is false, then the `authenticate_request!` method will call the `retrieve_current_user` method to retrieve the current user information.
- Once the current user information is retrieved, the `authenticate_request!` method will return true.
- If the `retrieve_current_user` method returns nil, then the `authenticate_request!` method will raise an exception.

JsonWebToken



```
1 require 'net/http'
2 require 'uri'
3
4 class JsonWebToken
5   attr_reader :token, :payload, :is_machine_to_machine
6
7   def initialize(token)
8     @token = token
9     @is_machine_to_machine = false
10 end
11
12 def verify
13   @payload = JWT.decode(
14     @token,
15     nil,
16     true, # Verify the signature of this token
17     algorithm: 'RS256',
18     verify_expiration: true,
19     iss: "#{ENV.fetch('MS__URL')}/",
20     verify_iss: true,
21     aud: ENV.fetch('TOKEN__AUDIENCE'),
22     verify_aud: true
23   ) { |header| jwks_hash(header['kid']) == header['kid'] }
24
25   @payload = @payload.try(:first)
26   @is_machine_to_machine = true unless @payload.key?('sub')
27   true
28 end
29
30 def jwks_hash(kid)
31   cached_jwks_keys = Rails.cache.fetch(kid, expires_in: 3.months) do
32     jwks_raw = Net::HTTP.get URI(
33       "#{ENV.fetch('TOKEN__JWKS_URL')}"
34     )
35     Array(JSON.parse(jwks_raw)['keys'])
36   end
37
38   cached_jwks_keys
39   .reject { |keyy| keyy['x5c'].blank? }
40   .to_h do |keyy|
41     [
42       keyy['kid'],
43       OpenSSL::X509::Certificate.new(
44         Base64.decode64(keyy['x5c'].first)
45       ).public_key
46     ]
47   end
48 end
49 end
50
```

Figure 5.5: Authentication - JsonWebToken

The JsonWebToken class is responsible for verifying the access token. It works as follows:

- The JsonWebToken class has a method called verify which is called by the verify_token method in the AuthorizationService class.
- The verify method first calls the jwks_hash method to get the public key of the Auth0 application that has been setup prior to the development of the STDC web application.
- The verify method then calls the JWT.decode method to decode the access token.

While decoding the access token, the JWT.decode method will verify the access token using the public key that has been retrieved from the jwks_hash method. If the access token is valid, then the JWT.decode method will return true, otherwise it will return false. For validation, the JWT.decode method will check the following:

- The signature of the access token.
 - The expiration of the access token.
 - The issuer of the access token.
 - The audience of the access token.
- If the JWT.decode method returns true, then the verify method will set the payload and is_machine_to_machine instance variables.
 - If the JWT.decode method returns false, then the verify method will raise an exception.

UsersService



```
1 require 'net/http'
2 require 'openssl'
3
4 class UsersController
5   attr_accessor :user_claims_data
6
7   def initialize(headers = {})
8     @access_token = headers['Authorization'].split.last if headers['Authorization'].present?
9     @uri = URI(ENV.fetch('USERINFO_URL'))
10    end
11
12  def retrieve_user_data(sub)
13    request_uri = @uri
14
15    http = Net::HTTP.new(@uri.host, @uri.port)
16    http.use_ssl = (@uri.scheme == 'https')
17    http.verify_mode = OpenSSL::SSL::VERIFY_NONE
18
19    request = Net::HTTP::Get.new(request_uri.to_s)
20
21    request['Authorization'] = "Bearer #{@access_token}"
22
23    response = http.request(request)
24
25    if response.code == '200'
26      @user_claims_data = JSON.parse(response.body)
27      return @user_claims_data if json_response['sub'] == sub
28    end
29
30    false
31  end
32 end
33
```

Figure 5.6: Authentication - UsersService

The UsersController class is responsible for retrieving the user information from Microsoft. It works as follows:

- The UsersController class has a method called retrieve_user_data which is called by the retrieve_current_user method in the AuthenticationService class.
- The retrieve_user_data first tries to fetch the API access token from the headers of the request.
- The retrieve_user_data then sends a request to Microsoft to get the user information to Microsoft's userinfo endpoint.
- If the request is successful, then the retrieve_user_data method will set the user information to current_user and then return true, else it will return false.

Now that all the classes and modules have been explained, we understand a few advantages of the explained authentication process:

- The authentication process is handled in the backend, which means that the client does not have to worry about the authentication process. Even if the client is compromised, the attacker will not be able to access the API without a valid access token.
- The usage of the OAuth 2.0 and OpenID Connect allows the STDC web application to be more secure, since the authentication process is handled by Auth0 and Microsoft.
- The API does not store user's credentials, which means that if the API were to be compromised, the attacker will not be able to access the user's credentials.
- The authentication process allows for multiple checks to be done to make sure that the access token is valid. If the access token has been tampered with, then the authentication process will fail since the signature of the access token will not match the signature of the public key obtained from the jwks of Auth0.
- Using a JWT access token gives the API the ability to verify the access token without having to make a request to Auth0 or Microsoft.

Authorization

The Student Talent Development Center (STDC) makes use of Cancancan to handle its authorization. Cancancan is an authorization library for Ruby on Rails which restricts what resources a given user is allowed to access. All permissions are defined in a single location (the Ability class) and not duplicated across controllers, views, and database queries. The following are the benefits of using Cancancan:

- Cancancan allows for the definition of permissions in a very simple and readable way.
- Cancancan allows for the definition of permissions in a very small and granular way.
- Cancancan allows for running queries on the database based on the permissions.

In the coming pages, we will discuss how Cancancan is used in the STDC web application.

In the STDC web application, the permissions are defined in the Ability class. The Ability class is a class defined by Cancancan, and it is responsible for defining the permissions. The Ability class is defined as follows:



```
1 class Ability
2   include CanCan::Ability
3   include Abilities::Tutor
4   include Abilities::Student
5   include Abilities::Viewer
6   include Abilities::Superadmin
7   include Abilities::M2m
8
9   def initialize(user)
10    user ||= User.new
11
12    if @access_is_machine_to_machine
13      m2m_abilities(user)
14    else
15      can :update, User, id: user.id
16
17      ROLES.each do |role|
18        send("#{role}_abilities", user) if user.role?(role)
19      end
20    end
21  end
22 end
23
```

Figure 5.7: Authorization - Ability

The Ability class works as follows:

- The Ability class has a method called initialize which is called by the Ability class itself.
- The initialize method first checks if the user is a machine to machine user or not.
- If the user is a machine to machine user, then the initialize method will call the m2m_abilities method to define the permissions for the machine to machine user.
- If the user is not a machine to machine user, then the initialize method will call the role_abilities method to define the permissions for the user based on his/her role.

- The initialize method will then return the permissions.

The CanCanCan gem has a method called can which is defined by Cancancan. The can method is used to define the permissions for a user. The can method takes three parameters, the first parameter is the action, the second parameter is the resource, and the third parameter is the conditions. The action parameter is the action that the user is allowed to perform on the resource. The resource parameter is the resource that the user is allowed to perform the action on. The conditions parameter is the conditions that the user is allowed to perform the action on the resource.

The Ability class has a method called m2m_abilities which is responsible for defining the permissions for the machine to machine user. The m2m_abilities method is defined as follows:



```

● ● ●

1 module Abilities
2   module M2m
3     def m2m_abilities(user)
4       can :list, :all if user.scope?('stdc.read_all')
5       can :read, :all if user.scope?('stdc.read')
6     end
7   end
8 end
9

```

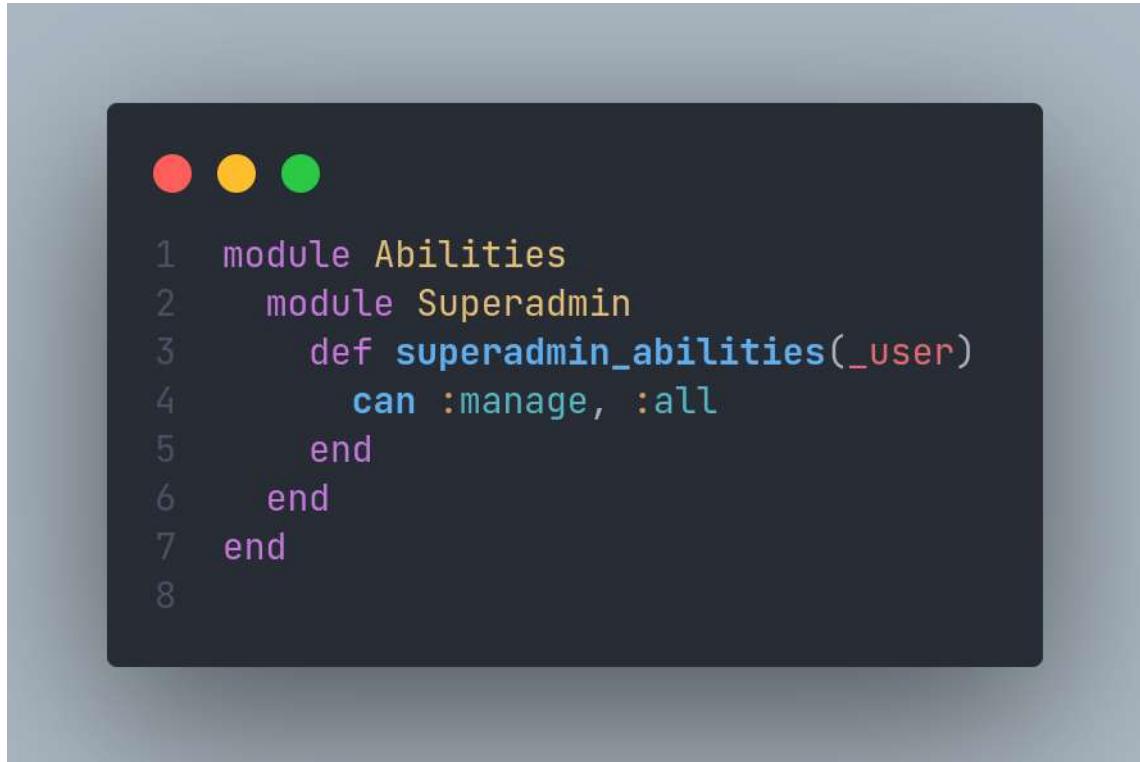
Figure 5.8: Authorization - m2m_abilities

The m2m_abilities method defines the permissions for the machine to machine user as follows:

- The machine to machine user can read any resource.
- The machine to machine user can list any resource.

The Ability class has a method called superadmin_abilities which is responsible for

defining the permissions for the superadmin user. The superadmin_abilities method is defined as follows:



```
● ● ●

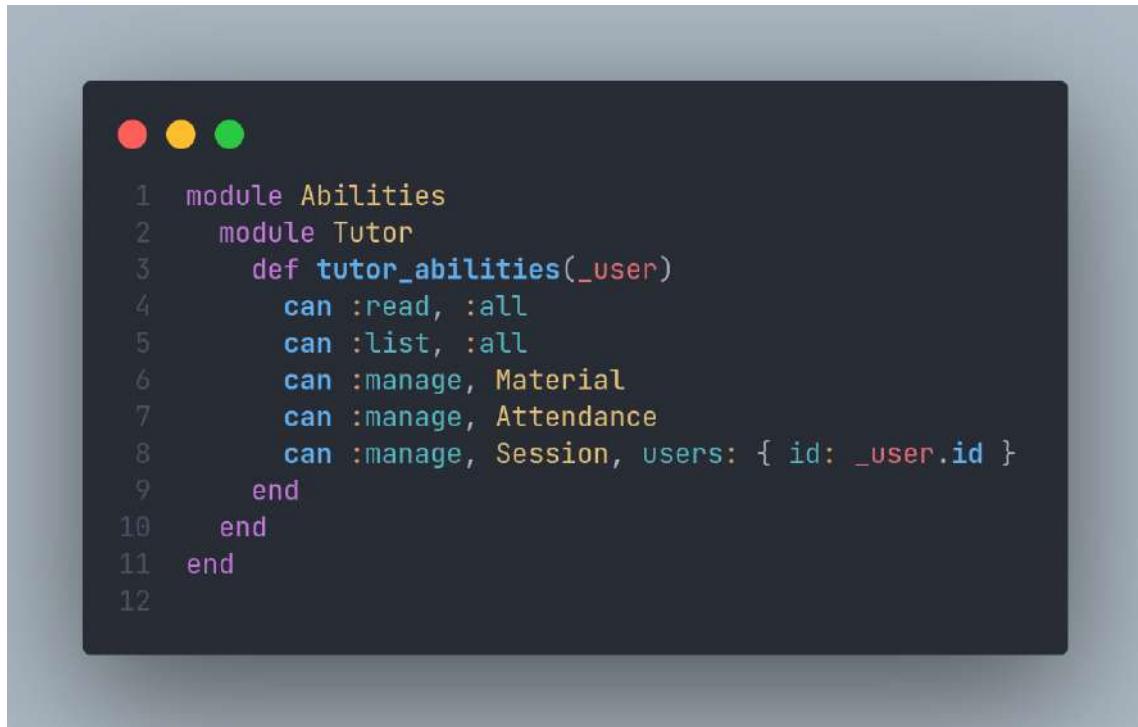
1 module Abilities
2   module Superadmin
3     def superadmin_abilities(_user)
4       can :manage, :all
5     end
6   end
7 end
8
```

Figure 5.9: Authorization - superadmin_abilities

The superadmin_abilities method defines the permissions for the superadmin user as follows:

- The superadmin can manage any resource.

The Ability class has a method called tutor_abilities which is responsible for defining the permissions for the tutor user. The tutor_abilities method is defined as follows:



```
1 module Abilities
2   module Tutor
3     def tutor_abilities(_user)
4       can :read, :all
5       can :list, :all
6       can :manage, Material
7       can :manage, Attendance
8       can :manage, Session, users: { id: _user.id }
9     end
10    end
11  end
12
```

Figure 5.10: Authorization - tutor_abilities

The tutor_abilities method defines the permissions for the tutor user as follows:

- The tutor can read any resource.
- The tutor can list any resource.
- The tutor can manage the material resource.
- The tutor can manage the attendance resource.
- The tutor can manage the session resource if he/she is the owner of the session.

The Ability class has a method called student_abilities which is responsible for defining the permissions for the student user. The student_abilities method is defined as follows:



```
1 module Abilities
2   module Student
3     def student_abilities(_user)
4       can :read, :all
5       can :list, :all
6       can :create, Feedback
7       can :create, Attendance
8     end
9   end
10 end
11
```

Figure 5.11: Authorization - student_abilities

The `student_abilities` method defines the permissions for the student user as follows:

- The student can read any resource.
- The student can list any resource.
- The student can create the feedback resource.
- The student can create the attendance resource.

The STDC web application's security when it comes to authorization has a few advantages, one of which is small granular permissions. The STDC web application's permissions are defined in a very small and granular way, which means that the permissions are defined for each action on each resource. This allows for more control over the permissions, and it allows for more flexibility when it comes to defining the

permissions. In case a user is compromised and his/her access token is stolen, the attacker will not be able to access the resources that the user is not authorized to access. This is because the permissions are defined in a very small and granular way, allows for minimal damage in case of a security breach.

Models

A model in the context of the Student Talent Development Center (STDC) web application typically encompasses various components that define its behavior and structure. These components include concerns, enums, validations, callbacks, associations, scopes, and methods. Each of these elements plays a crucial role in defining the characteristics and functionality of the model.

The Student Talent Development Center (STDC) application consists of multiple models, including answer, attendance, course, enrollment, feedback, material, question, session, and user. Each model shares a similar structure with the course (which will be explained in detail below) model and exhibits similar behaviors and functionalities.

Validations are used to ensure the integrity and validity of the data stored in a model. They define rules and constraints that must be met for the model to be considered valid. For example, the presence validation ensures that a specific attribute, such as the name of a course, is not empty. If the validation fails, the object will not be saved.

Enums are used to define a set of discrete values for an attribute. They allow for a more expressive and structured representation of certain attributes. In the context of the STDC application, enums are used to define values such as the status of a course (e.g., active or inactive). The actual values stored in the database are typically integers corresponding to the enum values, and Rails provides helper methods to convert between the integer and enum values.



Figure 5.12: Course Model - Enums

Methods in a model encapsulate business logic and define additional behaviors. They are used to perform calculations, retrieve related data, or implement custom functionality. For example, in the course model, there are methods like "tutor_name" to retrieve the name of the tutor for the course, "next_session_date" to determine the date of the next session, and "rating" to retrieve the rating of the course.



```

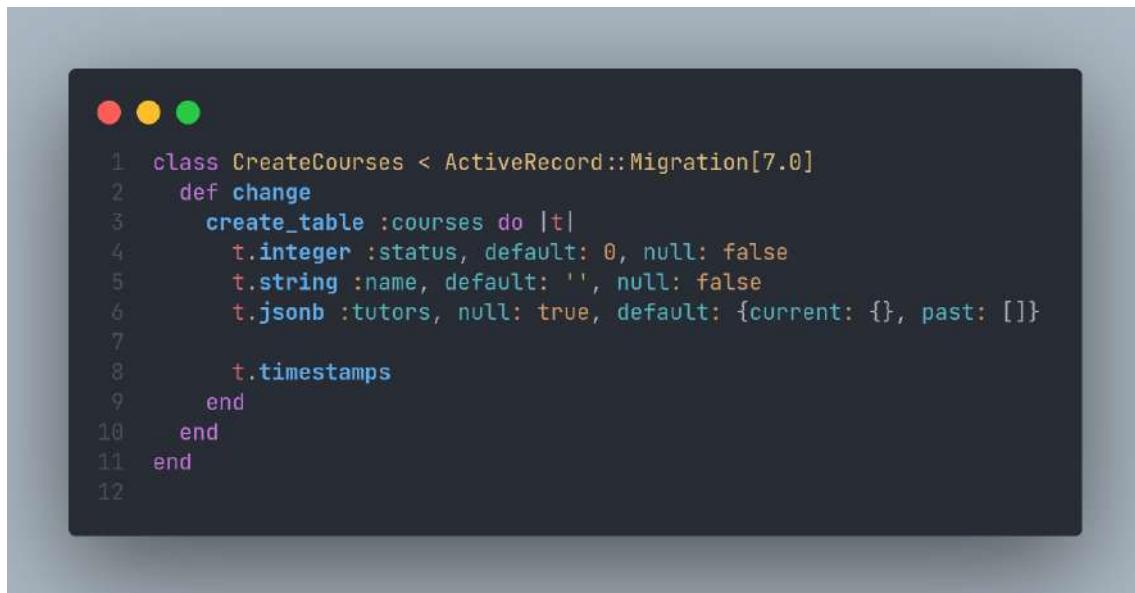
1 def tutor_name
2   tutors.try(:[], 'current').try(:[], 'name') || 'N/A'
3 end
4
5 def tutor_id
6   tutors.try(:[], 'current').try(:[], 'id') || 'N/A'
7 end
8
9 def next_session_date
10   date = Session.where(course: self).where('date > ? ', Date.today).order(:date).first.try(:date)
11   return "Next session on #{date.strftime('%d').to_i.ordinalize}" if date
12
13   'No upcoming sessions'
14 end
15
16 def next_session_venue_name
17   name = Venue.where(id: Session.where(course: self).where('date > ? ', Date.today).order(:date).first.try(:venue_id)).first.try(:name)
18   return "Next session at #{name}" if name
19
20   'No upcoming sessions'
21 end
22
23 def rating
24   sessions = Session.where(course: self).where.not(cancellation: 'cancelled')
25   return 0.0 if sessions.count.zero?
26
27   total_rating = sessions.map(&rating).sum
28   total_rating / (sessions.count.to_f - sessions.select { |s| s.rating.zero? }.count)
29 rescue StandardError => e
30   Rails.logger.error "Error while calculating rating for course: #{id}"
31   0.0
32 end
33
34 def update_tutor_metadata
35   return unless tutors.present? && tutors['current'].present?
36
37   user = nil
38
39   begin
40     user = User.find(tutors['current']['id'])
41   rescue StandardError => e
42     Rails.logger.error "User not found for id: #{tutors['current']['id']}"
43   end
44
45   return unless user.present?
46
47   tutors['current'] = user
48   Session.where(course_id: id).find_each do |session|
49     next unless session.date > Date.today
50
51     session.tutor = user
52
53     begin
54       Attendance.create(user_id: user.id, session_id: session.id, rating: 'five')
55     rescue StandardError => e
56       Rails.logger.error "Attendance already exists for user: #{user.id} and session: #{session.id}"
57     end
58
59     session.save
60   end
61 end
62 end

```

Figure 5.13: Course Model - Methods

Now, let's focus on the course model as an example for the rest of the section. The course model represents a specific course within the STDC application. It has attributes such as "name" (representing the name of the course), "status" (indicating the current status of the course), "tutors" (containing information about the course's tutors), and "created_at" and "updated_at" (representing timestamps for when the

course was created and last updated).



```
1 class CreateCourses < ActiveRecord::Migration[7.0]
2   def change
3     create_table :courses do |t|
4       t.integer :status, default: 0, null: false
5       t.string :name, default: '', null: false
6       t.jsonb :tutors, null: true, default: {current: {}, past: []}
7       t.timestamps
8     end
9   end
10 end
11 end
12
```

Figure 5.14: Course Model - Migration File

In terms of associations, the course model has several. It includes a "has_many :enrollments" association, establishing a one-to-many relationship with the Enrollment model. This allows a course to have multiple enrollments, and the "dependent: :destroy" option ensures that associated enrollments are destroyed when the course is destroyed. Additionally, the course model has associations with the User and Session models through enrollments.



```
1 # Associations
2 has_many :enrollments, dependent: :destroy
3 has_many :users, through: :enrollments
4 has_many :sessions, dependent: :destroy
```

Figure 5.15: Course Model - Associations

The course model also utilizes various validations to ensure data consistency. For instance, it includes a presence validation for the "name" attribute, guaranteeing that the name is not empty. Another validation, "validate_enum_attribute :status," ensures that the status is present in the model. These validations help maintain the integrity of the data stored in the course model.

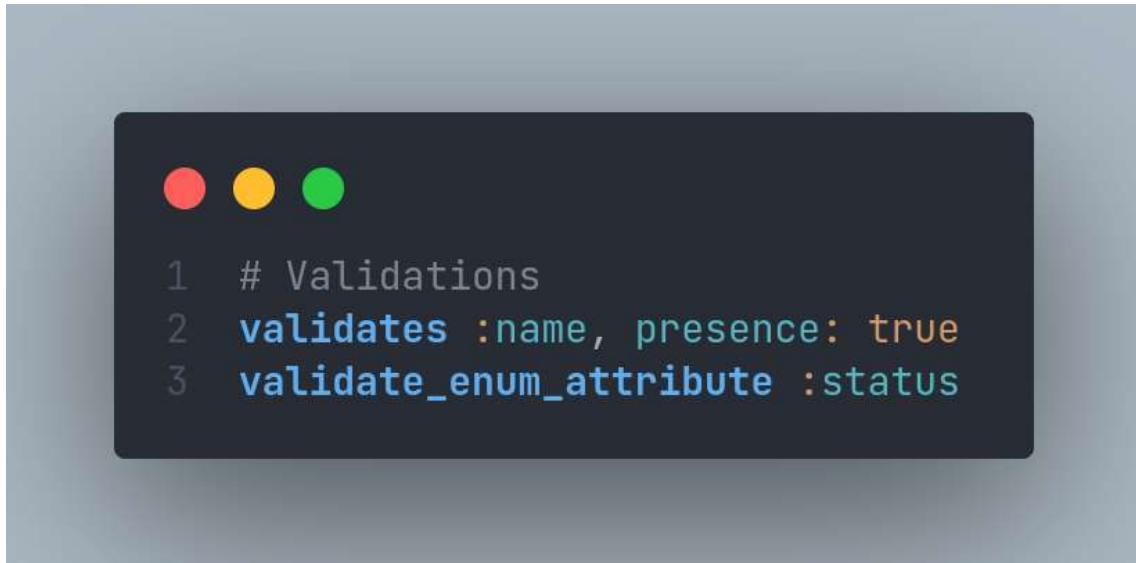


Figure 5.16: Course Model - Validations

Additionally, the course model employs callbacks, which are methods called at specific points in the model's lifecycle. For example, the model have an "after_save" callback that triggers the "update_tutor_metadata" method after the model is saved. Another callback, "after_commit :update_counter_cache, on: %i[create destroy]," updates the counter cache for the model, improving performance by storing the count of the model in the cache.



```
1 # Callbacks
2 after_save :update_tutor_metadata
```

Figure 5.17: Course Model - Callback



```
1 module CountCacheable
2   extend ActiveSupport::Concern
3
4   included do
5     after_commit :update_counter_cache, on: [:create, :destroy]
6
7   def update_counter_cache
8     Rails.cache.write("stdc_api_#{self.class.name.tableize.singularize}_count", self.class.count)
9   end
10 end
11 end
12
```

Figure 5.18: Course Model - Count Cachable Concern

Scopes in the course model provide named queries that allow for flexible and efficient filtering of model instances. For example, a "status" scope could filter courses by their status, while a "by_name" scope could filter courses by their name. These scopes simplify querying and provide a concise way to retrieve specific subsets of courses based on predefined criteria.



```
1 # Scopes
2 scope :status, lambda { |status|
3   value = statuses.keys.include?(status) ? status : statuses.keys.first
4   send(value)
5 }
6
7 scope :by_name, lambda { |name|
8   where('LOWER(courses.name) LIKE :q', q: "%#{name.to_s.downcase}%")
9 }
10
11 scope :by_user_id, lambda { |user_id|
12   joins(:enrollments).where(enrollments: { user_id: })
13 }
```

Figure 5.19: Course Model - Scopes

By utilizing these components such as validations, enums, methods, callbacks, associations, and scopes, the course model in the STDC application ensures data consistency, enables efficient querying and filtering, defines relationships with other models, and encapsulates essential behaviors and business logic.

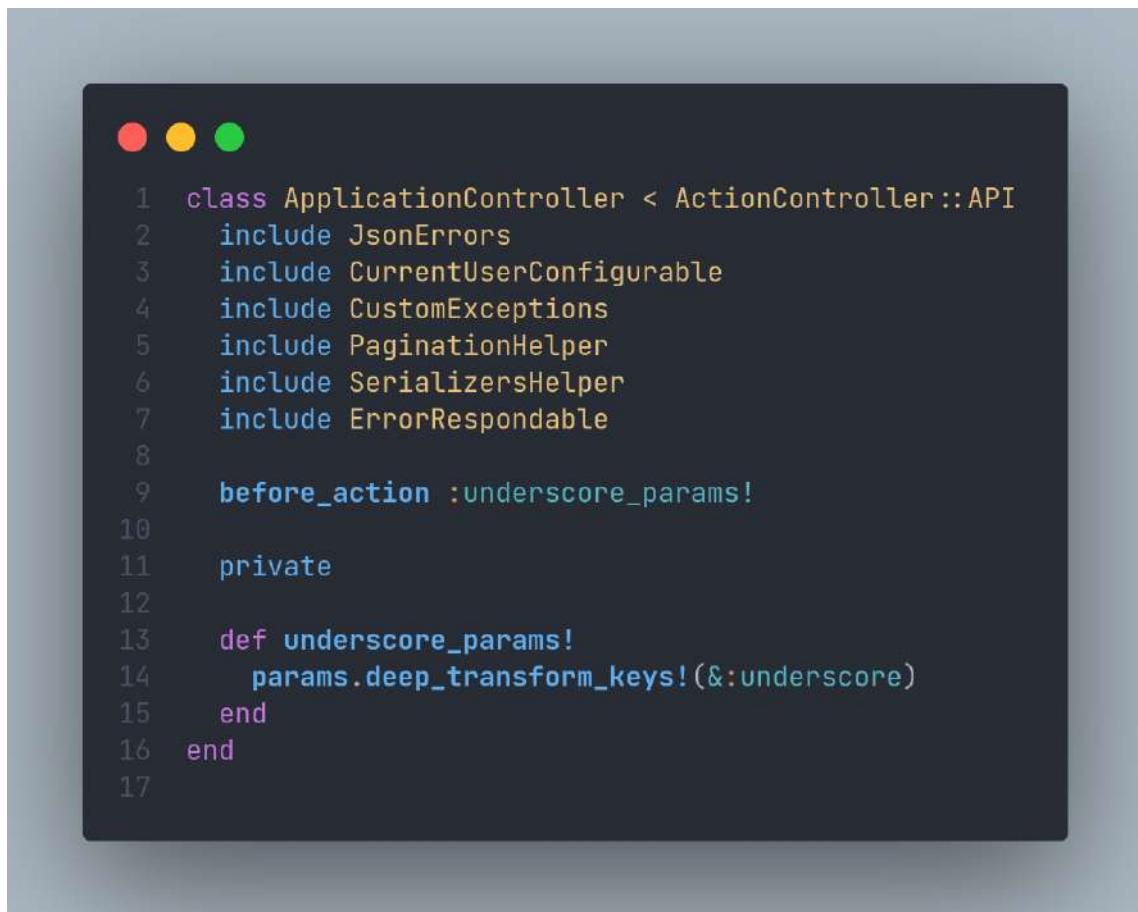
Controller

The following is a detailed explanation of the controllers used in the Student Talent Development Center (STDC) web application. Even though not all of the controllers will be mentioned, a thorough explanation of how one of the controllers work with their dependencies will be provided.

The STDC web application has a total of 10 controllers, which are as follows:

- ApplicationController
- GuardController
- AttendancesController
- CoursesController
- FeedbacksController
- MaterialsController
- QuestionsController
- SessionsController
- VenuesController
- UsersController

The ApplicationController is the base controller for all the other controllers. It is inherited by the GuardController which is the parent of all the other controllers. In the ApplicationController the following modules are included:



```
1 class ApplicationController < ActionController::API
2   include JsonErrors
3   include CurrentUserConfigurable
4   include CustomExceptions
5   include PaginationHelper
6   include SerializersHelper
7   include ErrorRespondable
8
9   before_action :underscore_params!
10
11  private
12
13  def underscore_params!
14    params.deep_transform_keys!(&:underscore)
15  end
16
17
```

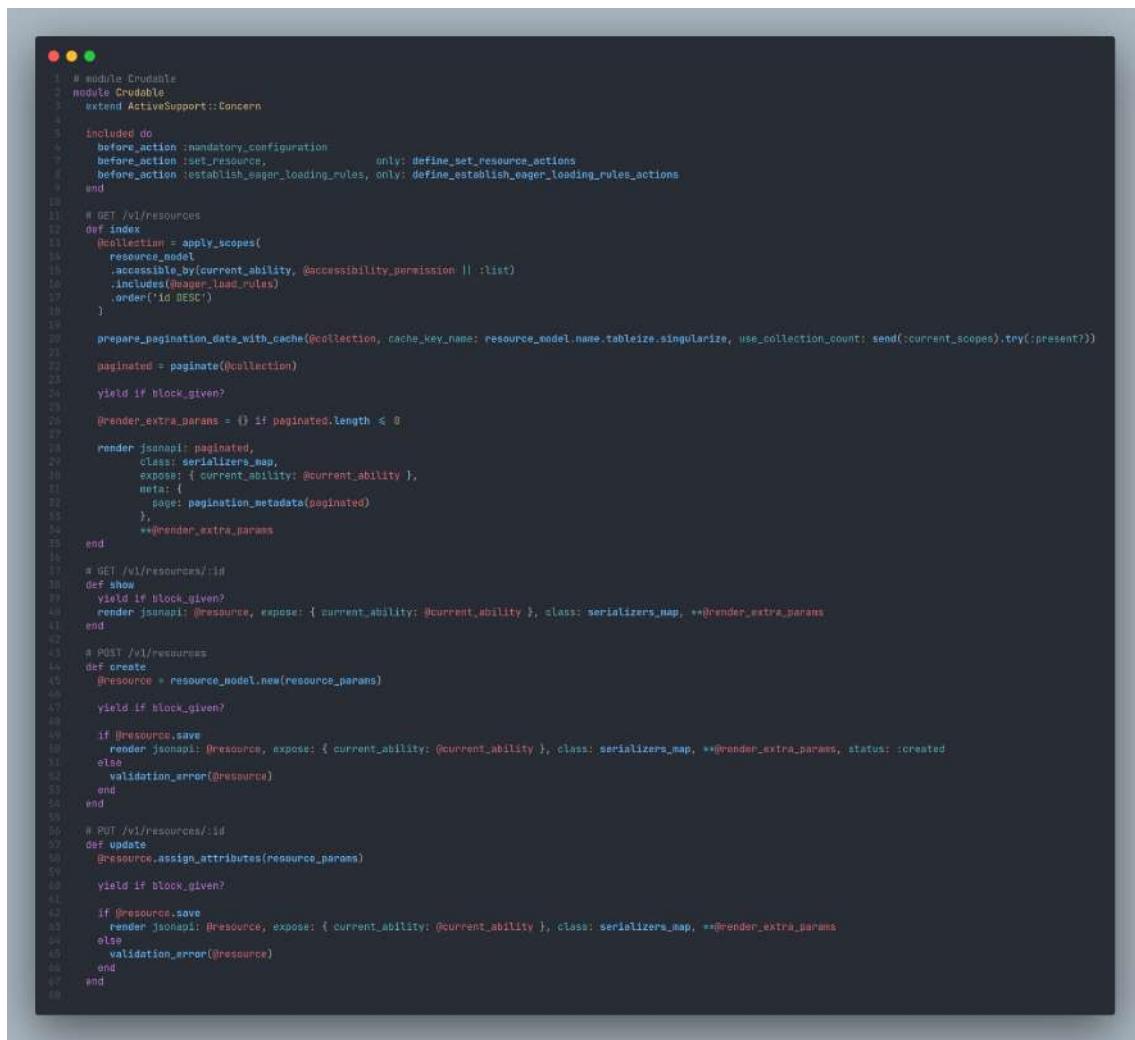
Figure 5.20: ApplicationController

- JsonErrors: This module is responsible for handling the errors that occur in the controllers.
- CurrentUserConfigurable: This module is responsible for configuring the current user by setting the current user after the user has been authenticated.
- CustomExceptions: This module is responsible for defining the custom exceptions that are used in the controllers.
- PaginationHelper: This module is responsible for handling the pagination of the resources.
- SerializersHelper: This module is responsible for handling the serialization of the resources.
- ErrorRespondable: This module is responsible for formatting the errors that occur in the controllers.

The ApplicationController has a method called underscore_params! which is responsible for transforming the keys of the params hash to underscore. This is done to ensure that the keys of the params hash are in the correct format of snake case before they are used in the controllers.

The GuardController has been mentioned prior to this section, therefore, we will not discuss it again. Now, let's dive into the Crudable and the CoursesController as an example.

The Crudable module is responsible for defining the CRUD actions for the controllers. The Crudable module is developed using Template Method design pattern. The Crudable module is included in the other controllers to define the CRUD actions for the controllers.

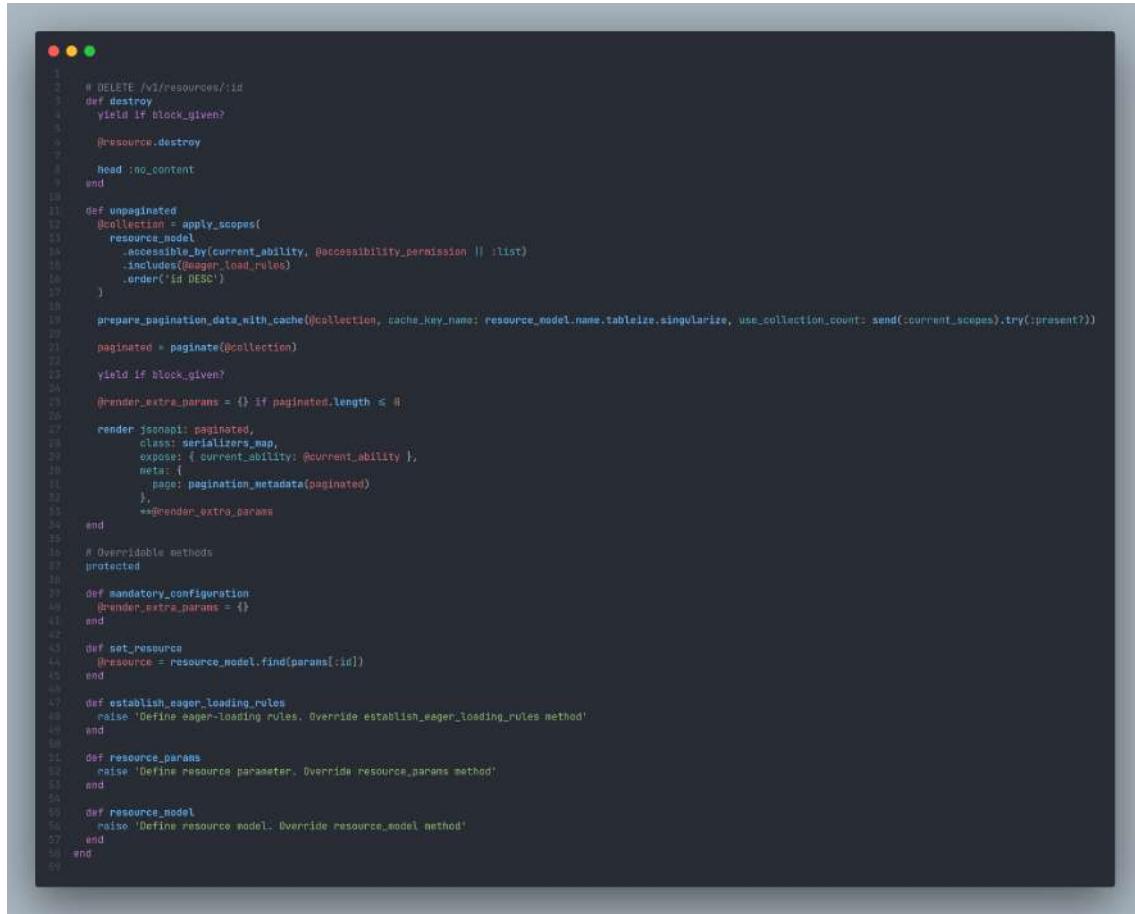


```

1  # module Crudable
2  module Crudable
3    extend ActiveSupport::Concern
4
5    included do
6      before_action :mandatory_configuration
7      before_action :set_resource, only: define_set_resource_actions
8      before_action :establish_eager_loading_rules, only: define_establish_eager_loading_rules_actions
9    end
10
11  # GET /v1/resources
12  def index
13    @collection = apply_scopes(
14      resource_model
15      .accessible_by(current_ability, @accessibility_permission || :list)
16      .includes(@inclusion_load_roles)
17      .order("id DESC")
18    )
19
20    prepare_pagination_data_with_cache(@collection, cache_key_name: resource_model.name.tableize.singularize, use_collection_count: send(:current_scopes).try(:present?))
21    paginated = paginate(@collection)
22
23    yield if block_given?
24
25    @render_extra_params = {} if paginated.length < 0
26
27    render jsonapi: paginated,
28          class: serializers_map,
29          expose: { current_ability: @current_ability },
30          meta: {
31            page: pagination_metadata(paginated)
32          },
33          **@render_extra_params
34  end
35
36  # GET /v1/resources/:id
37  def show
38    yield if block_given?
39    render jsonapi: @resource, expose: { current_ability: @current_ability }, class: serializers_map, **@render_extra_params
40  end
41
42  # POST /v1/resources
43  def create
44    @resource = resource_model.new(resource_params)
45
46    yield if block_given?
47
48    if @resource.save
49      render jsonapi: @resource, expose: { current_ability: @current_ability }, class: serializers_map, **@render_extra_params, status: :created
50    else
51      validation_error(@resource)
52    end
53  end
54
55  # PUT /v1/resources/:id
56  def update
57    @resource.assign_attributes(resource_params)
58
59    yield if block_given?
60
61    if @resource.save
62      render jsonapi: @resource, expose: { current_ability: @current_ability }, class: serializers_map, **@render_extra_params
63    else
64      validation_error(@resource)
65    end
66  end
67
68  # PATCH /v1/resources/:id
69  def patch
70    @resource.assign_attributes(resource_params)
71
72    yield if block_given?
73
74    if @resource.save
75      render jsonapi: @resource, expose: { current_ability: @current_ability }, class: serializers_map, **@render_extra_params
76    else
77      validation_error(@resource)
78    end
79  end
80
81  # DELETE /v1/resources/:id
82  def destroy
83    @resource.destroy
84
85    yield if block_given?
86
87    if @resource.destroyed?
88      head :no_content
89    else
90      validation_error(@resource)
91    end
92  end
93
94  # HEAD /v1/resources/:id
95  def head
96    @resource = resource_model.find(params[:id])
97
98    yield if block_given?
99
100   if @resource
101     head :ok
102   else
103     validation_error(@resource)
104   end
105  end
106
107  # GET /v1/resources/_count
108  def count
109    @count = resource_model.count
110
111    render jsonapi: { count: @count }
112  end
113
114  # GET /v1/resources/_meta
115  def meta
116    @meta = {
117      count: resource_model.count
118    }
119
120    render jsonapi: @meta
121  end
122
123  # GET /v1/resources/_table
124  def table
125    @table = resource_model.all
126
127    render jsonapi: @table
128  end
129
130  # GET /v1/resources/_index
131  def index
132    @index = resource_model.all
133
134    render jsonapi: @index
135  end
136
137  # GET /v1/resources/_show
138  def show
139    @show = resource_model.find(params[:id])
140
141    render jsonapi: @show
142  end
143
144  # GET /v1/resources/_create
145  def create
146    @create = resource_model.new(resource_params)
147
148    render jsonapi: @create
149  end
150
151  # GET /v1/resources/_update
152  def update
153    @update = resource_model.find(params[:id])
154
155    render jsonapi: @update
156  end
157
158  # GET /v1/resources/_patch
159  def patch
160    @patch = resource_model.find(params[:id])
161
162    render jsonapi: @patch
163  end
164
165  # GET /v1/resources/_destroy
166  def destroy
167    @destroy = resource_model.find(params[:id])
168
169    render jsonapi: @destroy
170  end
171
172  # GET /v1/resources/_head
173  def head
174    @head = resource_model.find(params[:id])
175
176    render jsonapi: @head
177  end
178
179  # GET /v1/resources/_meta
180  def meta
181    @meta = {
182      count: resource_model.count
183    }
184
185    render jsonapi: @meta
186  end
187
188  # GET /v1/resources/_table
189  def table
190    @table = resource_model.all
191
192    render jsonapi: @table
193  end
194
195  # GET /v1/resources/_index
196  def index
197    @index = resource_model.all
198
199    render jsonapi: @index
200  end
201
202  # GET /v1/resources/_show
203  def show
204    @show = resource_model.find(params[:id])
205
206    render jsonapi: @show
207  end
208
209  # GET /v1/resources/_create
210  def create
211    @create = resource_model.new(resource_params)
212
213    render jsonapi: @create
214  end
215
216  # GET /v1/resources/_update
217  def update
218    @update = resource_model.find(params[:id])
219
220    render jsonapi: @update
221  end
222
223  # GET /v1/resources/_patch
224  def patch
225    @patch = resource_model.find(params[:id])
226
227    render jsonapi: @patch
228  end
229
230  # GET /v1/resources/_destroy
231  def destroy
232    @destroy = resource_model.find(params[:id])
233
234    render jsonapi: @destroy
235  end
236
237  # GET /v1/resources/_head
238  def head
239    @head = resource_model.find(params[:id])
240
241    render jsonapi: @head
242  end
243
244  # GET /v1/resources/_meta
245  def meta
246    @meta = {
247      count: resource_model.count
248    }
249
250    render jsonapi: @meta
251  end
252
253  # GET /v1/resources/_table
254  def table
255    @table = resource_model.all
256
257    render jsonapi: @table
258  end
259
260  # GET /v1/resources/_index
261  def index
262    @index = resource_model.all
263
264    render jsonapi: @index
265  end
266
267  # GET /v1/resources/_show
268  def show
269    @show = resource_model.find(params[:id])
270
271    render jsonapi: @show
272  end
273
274  # GET /v1/resources/_create
275  def create
276    @create = resource_model.new(resource_params)
277
278    render jsonapi: @create
279  end
280
281  # GET /v1/resources/_update
282  def update
283    @update = resource_model.find(params[:id])
284
285    render jsonapi: @update
286  end
287
288  # GET /v1/resources/_patch
289  def patch
290    @patch = resource_model.find(params[:id])
291
292    render jsonapi: @patch
293  end
294
295  # GET /v1/resources/_destroy
296  def destroy
297    @destroy = resource_model.find(params[:id])
298
299    render jsonapi: @destroy
300  end
301
302  # GET /v1/resources/_head
303  def head
304    @head = resource_model.find(params[:id])
305
306    render jsonapi: @head
307  end
308
309  # GET /v1/resources/_meta
310  def meta
311    @meta = {
312      count: resource_model.count
313    }
314
315    render jsonapi: @meta
316  end
317
318  # GET /v1/resources/_table
319  def table
320    @table = resource_model.all
321
322    render jsonapi: @table
323  end
324
325  # GET /v1/resources/_index
326  def index
327    @index = resource_model.all
328
329    render jsonapi: @index
330  end
331
332  # GET /v1/resources/_show
333  def show
334    @show = resource_model.find(params[:id])
335
336    render jsonapi: @show
337  end
338
339  # GET /v1/resources/_create
340  def create
341    @create = resource_model.new(resource_params)
342
343    render jsonapi: @create
344  end
345
346  # GET /v1/resources/_update
347  def update
348    @update = resource_model.find(params[:id])
349
350    render jsonapi: @update
351  end
352
353  # GET /v1/resources/_patch
354  def patch
355    @patch = resource_model.find(params[:id])
356
357    render jsonapi: @patch
358  end
359
360  # GET /v1/resources/_destroy
361  def destroy
362    @destroy = resource_model.find(params[:id])
363
364    render jsonapi: @destroy
365  end
366
367  # GET /v1/resources/_head
368  def head
369    @head = resource_model.find(params[:id])
370
371    render jsonapi: @head
372  end
373
374  # GET /v1/resources/_meta
375  def meta
376    @meta = {
377      count: resource_model.count
378    }
379
380    render jsonapi: @meta
381  end
382
383  # GET /v1/resources/_table
384  def table
385    @table = resource_model.all
386
387    render jsonapi: @table
388  end
389
390  # GET /v1/resources/_index
391  def index
392    @index = resource_model.all
393
394    render jsonapi: @index
395  end
396
397  # GET /v1/resources/_show
398  def show
399    @show = resource_model.find(params[:id])
400
401    render jsonapi: @show
402  end
403
404  # GET /v1/resources/_create
405  def create
406    @create = resource_model.new(resource_params)
407
408    render jsonapi: @create
409  end
410
411  # GET /v1/resources/_update
412  def update
413    @update = resource_model.find(params[:id])
414
415    render jsonapi: @update
416  end
417
418  # GET /v1/resources/_patch
419  def patch
420    @patch = resource_model.find(params[:id])
421
422    render jsonapi: @patch
423  end
424
425  # GET /v1/resources/_destroy
426  def destroy
427    @destroy = resource_model.find(params[:id])
428
429    render jsonapi: @destroy
430  end
431
432  # GET /v1/resources/_head
433  def head
434    @head = resource_model.find(params[:id])
435
436    render jsonapi: @head
437  end
438
439  # GET /v1/resources/_meta
440  def meta
441    @meta = {
442      count: resource_model.count
443    }
444
445    render jsonapi: @meta
446  end
447
448  # GET /v1/resources/_table
449  def table
450    @table = resource_model.all
451
452    render jsonapi: @table
453  end
454
455  # GET /v1/resources/_index
456  def index
457    @index = resource_model.all
458
459    render jsonapi: @index
460  end
461
462  # GET /v1/resources/_show
463  def show
464    @show = resource_model.find(params[:id])
465
466    render jsonapi: @show
467  end
468
469  # GET /v1/resources/_create
470  def create
471    @create = resource_model.new(resource_params)
472
473    render jsonapi: @create
474  end
475
476  # GET /v1/resources/_update
477  def update
478    @update = resource_model.find(params[:id])
479
480    render jsonapi: @update
481  end
482
483  # GET /v1/resources/_patch
484  def patch
485    @patch = resource_model.find(params[:id])
486
487    render jsonapi: @patch
488  end
489
490  # GET /v1/resources/_destroy
491  def destroy
492    @destroy = resource_model.find(params[:id])
493
494    render jsonapi: @destroy
495  end
496
497  # GET /v1/resources/_head
498  def head
499    @head = resource_model.find(params[:id])
500
501    render jsonapi: @head
502  end
503
504  # GET /v1/resources/_meta
505  def meta
506    @meta = {
507      count: resource_model.count
508    }
509
510    render jsonapi: @meta
511  end
512
513  # GET /v1/resources/_table
514  def table
515    @table = resource_model.all
516
517    render jsonapi: @table
518  end
519
520  # GET /v1/resources/_index
521  def index
522    @index = resource_model.all
523
524    render jsonapi: @index
525  end
526
527  # GET /v1/resources/_show
528  def show
529    @show = resource_model.find(params[:id])
530
531    render jsonapi: @show
532  end
533
534  # GET /v1/resources/_create
535  def create
536    @create = resource_model.new(resource_params)
537
538    render jsonapi: @create
539  end
540
541  # GET /v1/resources/_update
542  def update
543    @update = resource_model.find(params[:id])
544
545    render jsonapi: @update
546  end
547
548  # GET /v1/resources/_patch
549  def patch
550    @patch = resource_model.find(params[:id])
551
552    render jsonapi: @patch
553  end
554
555  # GET /v1/resources/_destroy
556  def destroy
557    @destroy = resource_model.find(params[:id])
558
559    render jsonapi: @destroy
560  end
561
562  # GET /v1/resources/_head
563  def head
564    @head = resource_model.find(params[:id])
565
566    render jsonapi: @head
567  end
568
569  # GET /v1/resources/_meta
570  def meta
571    @meta = {
572      count: resource_model.count
573    }
574
575    render jsonapi: @meta
576  end
577
578  # GET /v1/resources/_table
579  def table
580    @table = resource_model.all
581
582    render jsonapi: @table
583  end
584
585  # GET /v1/resources/_index
586  def index
587    @index = resource_model.all
588
589    render jsonapi: @index
590  end
591
592  # GET /v1/resources/_show
593  def show
594    @show = resource_model.find(params[:id])
595
596    render jsonapi: @show
597  end
598
599  # GET /v1/resources/_create
600  def create
601    @create = resource_model.new(resource_params)
602
603    render jsonapi: @create
604  end
605
606  # GET /v1/resources/_update
607  def update
608    @update = resource_model.find(params[:id])
609
610    render jsonapi: @update
611  end
612
613  # GET /v1/resources/_patch
614  def patch
615    @patch = resource_model.find(params[:id])
616
617    render jsonapi: @patch
618  end
619
620  # GET /v1/resources/_destroy
621  def destroy
622    @destroy = resource_model.find(params[:id])
623
624    render jsonapi: @destroy
625  end
626
627  # GET /v1/resources/_head
628  def head
629    @head = resource_model.find(params[:id])
630
631    render jsonapi: @head
632  end
633
634  # GET /v1/resources/_meta
635  def meta
636    @meta = {
637      count: resource_model.count
638    }
639
640    render jsonapi: @meta
641  end
642
643  # GET /v1/resources/_table
644  def table
645    @table = resource_model.all
646
647    render jsonapi: @table
648  end
649
650  # GET /v1/resources/_index
651  def index
652    @index = resource_model.all
653
654    render jsonapi: @index
655  end
656
657  # GET /v1/resources/_show
658  def show
659    @show = resource_model.find(params[:id])
660
661    render jsonapi: @show
662  end
663
664  # GET /v1/resources/_create
665  def create
666    @create = resource_model.new(resource_params)
667
668    render jsonapi: @create
669  end
670
671  # GET /v1/resources/_update
672  def update
673    @update = resource_model.find(params[:id])
674
675    render jsonapi: @update
676  end
677
678  # GET /v1/resources/_patch
679  def patch
680    @patch = resource_model.find(params[:id])
681
682    render jsonapi: @patch
683  end
684
685  # GET /v1/resources/_destroy
686  def destroy
687    @destroy = resource_model.find(params[:id])
688
689    render jsonapi: @destroy
690  end
691
692  # GET /v1/resources/_head
693  def head
694    @head = resource_model.find(params[:id])
695
696    render jsonapi: @head
697  end
698
699  # GET /v1/resources/_meta
700  def meta
701    @meta = {
702      count: resource_model.count
703    }
704
705    render jsonapi: @meta
706  end
707
708  # GET /v1/resources/_table
709  def table
710    @table = resource_model.all
711
712    render jsonapi: @table
713  end
714
715  # GET /v1/resources/_index
716  def index
717    @index = resource_model.all
718
719    render jsonapi: @index
720  end
721
722  # GET /v1/resources/_show
723  def show
724    @show = resource_model.find(params[:id])
725
726    render jsonapi: @show
727  end
728
729  # GET /v1/resources/_create
730  def create
731    @create = resource_model.new(resource_params)
732
733    render jsonapi: @create
734  end
735
736  # GET /v1/resources/_update
737  def update
738    @update = resource_model.find(params[:id])
739
740    render jsonapi: @update
741  end
742
743  # GET /v1/resources/_patch
744  def patch
745    @patch = resource_model.find(params[:id])
746
747    render jsonapi: @patch
748  end
749
750  # GET /v1/resources/_destroy
751  def destroy
752    @destroy = resource_model.find(params[:id])
753
754    render jsonapi: @destroy
755  end
756
757  # GET /v1/resources/_head
758  def head
759    @head = resource_model.find(params[:id])
760
761    render jsonapi: @head
762  end
763
764  # GET /v1/resources/_meta
765  def meta
766    @meta = {
767      count: resource_model.count
768    }
769
770    render jsonapi: @meta
771  end
772
773  # GET /v1/resources/_table
774  def table
775    @table = resource_model.all
776
777    render jsonapi: @table
778  end
779
780  # GET /v1/resources/_index
781  def index
782    @index = resource_model.all
783
784    render jsonapi: @index
785  end
786
787  # GET /v1/resources/_show
788  def show
789    @show = resource_model.find(params[:id])
790
791    render jsonapi: @show
792  end
793
794  # GET /v1/resources/_create
795  def create
796    @create = resource_model.new(resource_params)
797
798    render jsonapi: @create
799  end
800
801  # GET /v1/resources/_update
802  def update
803    @update = resource_model.find(params[:id])
804
805    render jsonapi: @update
806  end
807
808  # GET /v1/resources/_patch
809  def patch
810    @patch = resource_model.find(params[:id])
811
812    render jsonapi: @patch
813  end
814
815  # GET /v1/resources/_destroy
816  def destroy
817    @destroy = resource_model.find(params[:id])
818
819    render jsonapi: @destroy
820  end
821
822  # GET /v1/resources/_head
823  def head
824    @head = resource_model.find(params[:id])
825
826    render jsonapi: @head
827  end
828
829  # GET /v1/resources/_meta
830  def meta
831    @meta = {
832      count: resource_model.count
833    }
834
835    render jsonapi: @meta
836  end
837
838  # GET /v1/resources/_table
839  def table
840    @table = resource_model.all
841
842    render jsonapi: @table
843  end
844
845  # GET /v1/resources/_index
846  def index
847    @index = resource_model.all
848
849    render jsonapi: @index
850  end
851
852  # GET /v1/resources/_show
853  def show
854    @show = resource_model.find(params[:id])
855
856    render jsonapi: @show
857  end
858
859  # GET /v1/resources/_create
860  def create
861    @create = resource_model.new(resource_params)
862
863    render jsonapi: @create
864  end
865
866  # GET /v1/resources/_update
867  def update
868    @update = resource_model.find(params[:id])
869
870    render jsonapi: @update
871  end
872
873  # GET /v1/resources/_patch
874  def patch
875    @patch = resource_model.find(params[:id])
876
877    render jsonapi: @patch
878  end
879
880  # GET /v1/resources/_destroy
881  def destroy
882    @destroy = resource_model.find(params[:id])
883
884    render jsonapi: @destroy
885  end
886
887  # GET /v1/resources/_head
888  def head
889    @head = resource_model.find(params[:id])
890
891    render jsonapi: @head
892  end
893
894  # GET /v1/resources/_meta
895  def meta
896    @meta = {
897      count: resource_model.count
898    }
899
900    render jsonapi: @meta
901  end
902
903  # GET /v1/resources/_table
904  def table
905    @table = resource_model.all
906
907    render jsonapi: @table
908  end
909
910  # GET /v1/resources/_index
911  def index
912    @index = resource_model.all
913
914    render jsonapi: @index
915  end
916
917  # GET /v1/resources/_show
918  def show
919    @show = resource_model.find(params[:id])
920
921    render jsonapi: @show
922  end
923
924  # GET /v1/resources/_create
925  def create
926    @create = resource_model.new(resource_params)
927
928    render jsonapi: @create
929  end
930
931  # GET /v1/resources/_update
932  def update
933    @update = resource_model.find(params[:id])
934
935    render jsonapi: @update
936  end
937
938  # GET /v1/resources/_patch
939  def patch
940    @patch = resource_model.find(params[:id])
941
942    render jsonapi: @patch
943  end
944
945  # GET /v1/resources/_destroy
946  def destroy
947    @destroy = resource_model.find(params[:id])
948
949    render jsonapi: @destroy
950  end
951
952  # GET /v1/resources/_head
953  def head
954    @head = resource_model.find(params[:id])
955
956    render jsonapi: @head
957  end
958
959  # GET /v1/resources/_meta
960  def meta
961    @meta = {
962      count: resource_model.count
963    }
964
965    render jsonapi: @meta
966  end
967
968  # GET /v1/resources/_table
969  def table
970    @table = resource_model.all
971
972    render jsonapi: @table
973  end
974
975  # GET /v1/resources/_index
976  def index
977    @index = resource_model.all
978
979    render jsonapi: @index
980  end
981
982  # GET /v1/resources/_show
983  def show
984    @show = resource_model.find(params[:id])
985
986    render jsonapi: @show
987  end
988
989  # GET /v1/resources/_create
990  def create
991    @create = resource_model.new(resource_params)
992
993    render jsonapi: @create
994  end
995
996  # GET /v1/resources/_update
997  def update
998    @update = resource_model.find(params[:id])
999
1000   render jsonapi: @update
1001 end

```

Figure 5.21: Crudable - Part 1



```
1  # DELETE /v1/resources/:id
2  def destroy
3    yield if block_given?
4
5    @resource.destroy
6
7    head :no_content
8  end
9
10 def unpaginated
11   @collection = apply_scopes(
12     resource_model
13     .accessible_by(current_ability, :accessibility_permission || :list)
14     .includes(:eager_load_rules)
15     .order('id DESC')
16   )
17
18   prepare_pagination_data_with_cache(@collection, cache_key_name: resource_model.name.tableize.singularize, use_collection_count: send(:current_scopes).try(:present?))
19
20   paginated = paginate(@collection)
21
22   yield if block_given?
23
24   @render_extra_params = {} if paginated.length < 8
25
26   render jsonapi: paginated,
27         class: serializers_map,
28         expose: { current_ability: @current_ability },
29         meta: {
30           page: pagination_metadata(paginated)
31         },
32         **@render_extra_params
33   end
34
35   # Overridable methods
36   protected
37
38   def mandatory_configuration
39     @render_extra_params = {}
40   end
41
42   def set_resource
43     @resource = resource_model.find(params[:id])
44   end
45
46   def establish_eager_loading_rules
47     raise 'Define eager-loading rules. override establish_eager_loading_rules method'
48   end
49
50   def resource_params
51     raise 'Define resource parameter. Override resource_params method'
52   end
53
54   def resource_model
55     raise 'Define resource model. Override resource_model method'
56   end
57
58 end
```

Figure 5.22: Crudable - Part 2

The Crudable requires the following methods to be defined in the controllers that include it:

- **mandatory_configuration**: This method is responsible for defining the mandatory configuration for the controller.
- **set_resource**: This method is responsible for setting the resource for the controller. Finding single resource to be used by other methods such as show and update.
- **establish_eager_loading_rules**: This method is responsible for establishing the eager loading rules for the controller. Eager loading is a concept in Rails that allows you to load all the associated objects of the object you are querying in a single query. This is done to avoid the N+1 query problem.
- **resource_params**: This method is responsible for defining the resource param-

eters for the controller. The resource parameters are the parameters that are allowed to be passed to the controller. It is a way to whitelist the parameters that are allowed to be passed to the controller which is a security measure.

- `resource_model`: This method is responsible for defining the resource model for the controller. The resource model is the model that the controller is responsible for. For example, the `CoursesController` is responsible for the `Course` model.

The `Crudable` also makes use of `PaginationHelper` and `SerializersHelper` modules. The `PaginationHelper` module is responsible for handling the pagination of the resources. The `SerializersHelper` module is responsible for handling the serialization of the resources. Both of these will map the response to the JSON API specification. More on each will be discussed later. Now, let's take a look at the `CoursesController`.



```
module PaginationHelper
  def pagination_metadata(collection, model_name: '')
    count = @optimized_paginated_count || Rails.cache.read("stdc_api_#{model_name}_count") || 0
    {
      totalPages: collection.try(:total_pages) || 1,
      count,
      limitValue: collection.try(:limit_value) || count,
      currentPage: collection.try(:current_page) || 1
    }
  end
  def prepare_pagination_data_with_cache(collection, cache_key_name:, use_collection_count: false)
    count = @optimized_paginated_count = use_collection_count ? collection.count : Rails.cache.read("stdc_api_#{cache_key_name}_count") || 0
    # Alter the total_count to use the count of choice -- cached
    collection.define_singleton_method(:total_count) { count.to_i }
  end
end
```

Figure 5.23: `PaginationHelper`

The `CoursesController` is responsible for handling the requests related to the `Course` model. The `CoursesController` includes the `Crudable` module, and it defines the basic CRUD actions for the `Course` model. The `CoursesController` also has a few more endpoints defined in it. The `CoursesController` has the following endpoints defined in it:

- GET `/v1/courses`: This endpoint is responsible for listing the courses, it is a paginated endpoint which means that it can be filtered, sorted, and paginated.
- GET `/v1/courses/:id`: This endpoint is responsible for showing a specific course.

- POST /v1/courses: This endpoint is responsible for creating a new course.
- PUT /v1/courses/:id: This endpoint is responsible for updating a specific course.
- DELETE /v1/courses/:id: This endpoint is responsible for deleting a specific course.
- GET /v1/courses/unpaginated: This endpoint is responsible for listing the courses, it is an unpaginated endpoint which means that it cannot be filtered and sorted.
- GET /v1/courses/:id/students: This endpoint is responsible for listing the students of a specific course.
- POST /v1/courses/assign_tutor: This endpoint is responsible for assigning a tutor to a specific course.

The CoursesController overrides the create and update methods defined in the Crudable module. The create and update methods are overridden to assign a tutor to the course if the user_id parameter is passed to the controller.

```
1 module V1
2   class CoursesController < GuardController
3     def self.define_set_resource_actions
4       %i[show update students]
5     end
6
7     def self.define_establish_eager_loading_rules_actions
8       %i[index unpaginated]
9     end
10
11    def establish_eager_loading_rules
12      @eager_load_rules = %i[]
13    end
14
15    def resource_model
16      Course
17    end
18
19    has_scope :status, as: :status, only: %i[index unpaginated]
20    has_scope :by_name, as: :name, only: %i[index unpaginated]
21    has_scope :by_user_id, as: :user_id, only: %i[index unpaginated]
22
23    include Crudable
24
25    def index
26      authorize! :list, resource_model
27
28      @render_extra_params = { include: %i[] }
29      super
30    end
31
32    def show
33      authorize! :read, @resource
34
35      @render_extra_params = { include: [] }
36
37      super
38    end
39
40    def create
41      authorize! :create, resource_model
42
43      @render_extra_params = { include: [] }
44
45      assign_tutor_to_course = proc do
46        if params[:data][:user_id].present?
47          Enrollment.create(user_id: params[:data][:user_id], course_id: @resource.id)
48          @resource.tutors = {current: {}, past: []} if @resource.tutors.blank?
49          @resource.tutors['current'] = User.find(params[:data][:user_id])
50          @resource.save
51        end
52      end
53
54      super(&assign_tutor_to_course)
55    end
56
```

Figure 5.24: CoursesController - Part 1

```

1   def update
2     authorize! :update, @resource
3
4     @render_extra_params = { include: [] }
5
6     assign_tutor_to_course = proc do
7       if params[:data][:user_id].present?
8         Enrollment.create(user_id: params[:data][:user_id], course_id: @resource.id)
9         @resource.tutors = {current: {}, past: []} if @resource.tutors.blank?
10
11       if !@resource.tutors['current'].present?
12         @resource.tutors['current'] = User.find(params[:data][:user_id])
13       else
14         @resource.tutors['past'] << @resource.tutors['current']
15         @resource.tutors['current'] = User.find(params[:data][:user_id])
16       end
17
18       @resource.save
19     end
20   end
21
22   super(&assign_tutor_to_course)
23 end
24
25
26 def unpaginated
27   authorize! :list, resource_model
28
29   @render_extra_params = { include: [] }
30
31   super
32 end
33
34 def students
35   authorize! :read, @resource
36
37   students = @resource.users.by_roles('student')
38
39   render jsonapi: students, expose: { current_ability: @current_ability }, class: serializers_map
40 end
41
42 def assign_tutor
43   @resource = resource_model.find(params[:data][:course_id])
44   @user = User.find(params[:data][:user_id])
45
46   authorize! :update, @resource
47
48   @resource.tutors = {current: {}, past: []} if @resource.tutors.blank?
49
50   if !@resource.tutors['current'].present?
51     @resource.tutors['current'] = @user
52   else
53     @resource.tutors['past'] << @resource.tutors['current']
54     @resource.tutors['current'] = @user
55   end
56
57   Enrollment.create(user_id: @user&.id, course_id: @resource&.id)
58
59   @resource.save
60
61   render jsonapi: @resource, expose: { current_ability: @current_ability }, class: serializers_map
62 end
63
64 private
65
66 def resource_params
67   params.require(:data).permit(
68     :name,
69     :status,
70     :tutors
71   )
72 end
73 end
74 end
75

```

Figure 5.25: CoursesController - Part 2

Serializations

The STDC web application uses the JSON API specification for serializing the resources. The JSON API specification is a specification for building APIs in JSON. It defines a set of rules for how the resources should be represented in JSON. The JSON API specification is used to ensure that the resources are represented in a consistent way across all the endpoints. In the STDC web application, each model comes with a serializer, that serializer uses the JSONAPI-rails gem to define what should be the out of an endpoint that returns a resource of that model. The JSONAPI-rails gem is a gem that provides a DSL for defining the JSON API specification.

Each serializer inherits from the BaseSerializer, the BaseSerializer is responsible for camelizing all the keys of a serialized resource. This is another way to standardize the responses of each endpoint. The reason for using Camel Case as response is that the client of the STDC uses React JS and the convention in React JS is to use Camel Case for the keys of the JSON objects.

The BaseSerializer is defined as follows:

A screenshot of a terminal window with a dark background. At the top left are three colored window control buttons (red, yellow, green). The terminal window contains the following Ruby code:

```
1 class BaseSerializer < JSONAPI::Serializable::Resource
2   extend JSONAPI::Serializable::Resource::KeyFormat
3
4   key_format { |key| key.to_s.camelize :lower }
5 end
6
```

Figure 5.26: BaseSerializer

The SerializersHelper is module that maps all the models to their corresponding serializers. The SerializersHelper is defined as follows:

```
1 module SerializersHelper
2   def serializers_map
3     {
4       User: V1::UserSerializer,
5       Course: V1::CourseSerializer,
6       Venue: V1::VenueSerializer,
7       Session: V1::SessionSerializer,
8       Attendance: V1::AttendanceSerializer,
9       Feedback: V1::FeedbackSerializer,
10      Question: V1::QuestionSerializer,
11      Answer: V1::AnswerSerializer,
12      Material: V1::MaterialSerializer
13     }.freeze
14   end
15 end
16
```

Figure 5.27: SerializersHelper

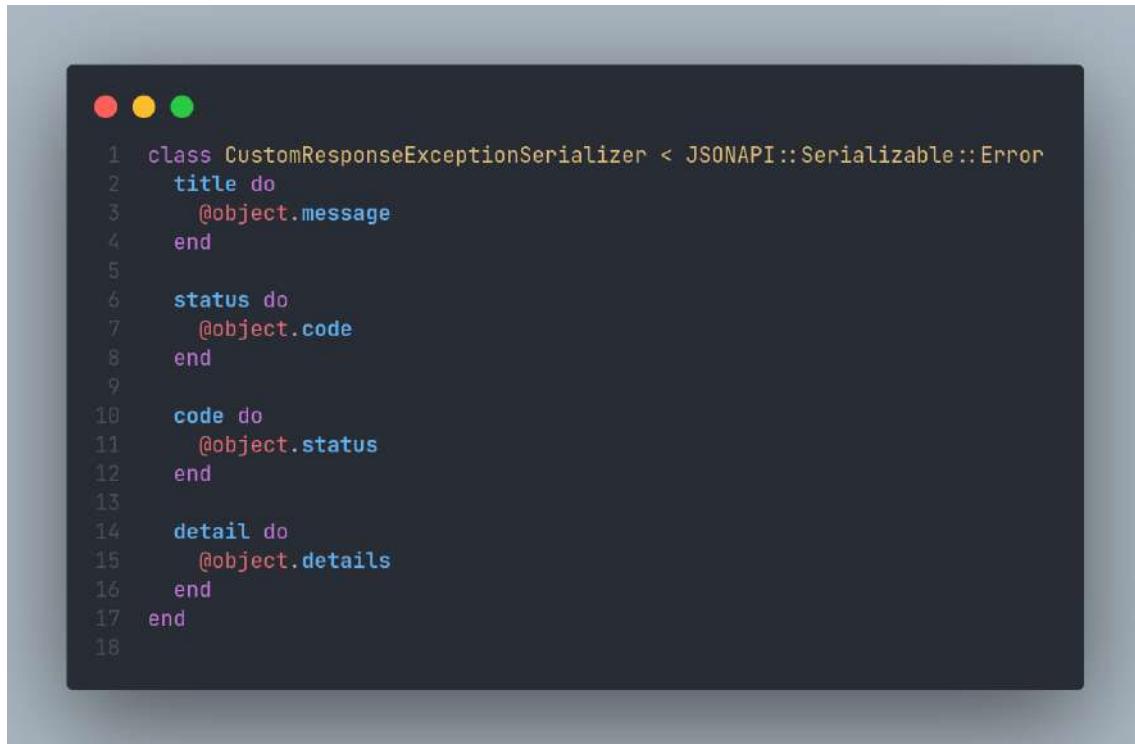
The following is an example of a serializer for the Course model:

```
1 module V1
2   class CourseSerializer < BaseSerializer
3     type 'courses'
4
5     attributes :id,
6                 :status,
7                 :name
8
9     attribute :rating do
10      @object.rating
11    end
12
13    attribute :tutor do
14      @object.tutor_name
15    end
16
17    attribute :tutor_id do
18      @object.tutor_id
19    end
20
21    attribute :next_session do
22      @object.next_session_date
23    end
24
25    attribute :next_session_venue do
26      @object.next_session_venue_name
27    end
28
29    attributes :created_at, :updated_at
30  end
31 end
32
```

Figure 5.28: CourseSerializer

Exception Management

The STDC web application uses the JSON API specification for serializing the resources this includes the errors. The STDC uses the CustomResponseExceptionSerializer to serialize the errors. The CustomResponseExceptionSerializer is defined as follows:



```
1 class CustomResponseExceptionSerializer < JSONAPI::Serializable::Error
2   title do
3     @object.message
4   end
5
6   status do
7     @object.code
8   end
9
10  code do
11    @object.status
12  end
13
14  detail do
15    @object.details
16  end
17 end
18
```

Figure 5.29: CustomResponseExceptionSerializer

The STDC web application uses the ErrorRespondable module to handle the errors that occur in the controllers. The ErrorRespondable module includes standard errors as well as custom errors. The standard errors are the errors that are defined by Rails itself. The custom errors are the errors that are defined in the CustomExceptions module and the JsonErrors. All the errors are handled in the ErrorRespondable in the same way.

```
1 module ErrorRespondable
2   extend ActiveSupport::Concern
3
4   standard_errors = [
5     { exception_class: ActionController::BadRequest, status: :bad_request, code: 400, message: 'Bad Request' },
6     { exception_class: ActiveRecord::RecordNotFound, status: :not_found, code: 404, message: 'Record was not found' },
7     { exception_class: CanCan::AccessDenied, status: :forbidden, code: 403 },
8     { exception_class: ActionController::ParameterMissing, status: :bad_request, code: 400 }
9   ]
10
11   included do
12     CustomExceptions::CustomResponseException.subclasses.each do |exception|
13       rescue_from exception_class do |exception|
14         render jsonapi_errors: exception, status: exception.status,
15               class: { exception.class.to_s.to_sym => CustomResponseExceptionSerializer }
16       end
17     end
18
19     standard_errors.each do |error|
20       rescue_from error[:exception_class] do |exception|
21         exception.define_singleton_method(:status) { error[:status] }
22         exception.define_singleton_method(:code) { error[:code] }
23         exception.define_singleton_method(:message) { error[:message] } if error.key?(:message)
24         exception.define_singleton_method(:details) { '' }
25
26         render jsonapi_errors: exception, status: error[:status],
27               class: { error[:exception_class].to_s.to_sym => CustomResponseExceptionSerializer }
28       end
29     end
30
31     def validation_error(resource)
32       resource.errors.send(:define_singleton_method, :keys) do
33         resource.errors.map(&:attribute)
34       end
35
36       render jsonapi_errors: resource.errors, status: :unprocessable_entity, code: 422
37     end
38   end
39 end
40
```

Figure 5.30: ErrorRespondable

```
1 module JsonErrors
2   extend ActiveSupport::Concern
3
4   included do
5     rescue_from ActiveRecord::RecordNotFound do |exception|
6       render json: { status: :not_found, code: 404, errors: exception.message.downcase }, status: :not_found,
7               code: 404
8     end
9
10    rescue_from CanCan::AccessDenied do |exception|
11      render json: { status: :forbidden, code: 403, errors: exception.message.downcase }, status: :forbidden,
12               code: 403
13    end
14
15    rescue_from CustomExceptions::UnauthorizedAccessException do |exception|
16      render json: { status: :unauthorized, code: 401, errors: exception.message.downcase }, status: :unauthorized,
17               code: 401
18    end
19  end
20 end
21
```

Figure 5.31: JsonErrors

```
1 module CustomExceptions
2   class CustomResponseException < StandardError; end
3
4   class UnauthorizedAccessException < CustomResponseException
5     def code
6       401
7     end
8
9     def status
10    :unauthorized
11  end
12
13   def details
14   ''
15  end
16
17   def message
18   'You are not authorized'
19  end
20 end
21
22 class UnprocessableEntityError < CustomResponseException
23   attr_reader :field, :details
24
25   def initialize(field:, message:)
26     super
27     @field = field
28     @details = message
29   end
30
31   def code
32     422
33   end
34
35   def status
36     :unprocessable_entity
37   end
38
39   def message
40   "Invalid #{@field}"
41  end
42 end
43 end
44
```

Figure 5.32: CustomExceptions

The Common Error Response HTTP Codes are as follows:

- 400: Bad Request (The request was invalid or cannot be served. The exact error is explained in the error payload).
- 401: Unauthorized (The request requires an user authentication).
- 403: Forbidden (The server understood the request, but is refusing it or the access is not allowed, requires user authorization).
- 404: Not Found (There is no resource behind the URI).
- 422: Unprocessable Entity (The request was well-formed but was unable to be followed due to semantic errors or invalid payload structure).
- 500: Internal Server Error (The server encountered an unexpected condition which prevented it from fulfilling the request).

There are more to the controllers than what has been mentioned in this section, however, the mentioned parts are the most important parts of the controllers. The other parts of the controllers are not mentioned in this section because they are not as important as the mentioned parts and due to the documentation size limitation.

Results

The Student Talent Development Center (STDC)'s backend has been implemented using Ruby on Rails. The development of the backend has been done to serve as an API server for the frontend. The backend has also been deployed to Heroku to be used by the frontend, more on this in the coming sections. The backend has no user interface, only API endpoints. However, the backend has API documentations that are generated using Swagger. The following are the API endpoints that are available in the backend of the STDC web application. They are taken from the Swagger API documentations from the deployed backend.

The screenshot shows the Swagger UI interface for an API V1 definition. At the top, there's a navigation bar with the Swagger logo, a dropdown for 'Select a definition' set to 'API V1 Docs', and a dropdown for 'Supported by SMARTBEAR'. Below the header, the title 'API V1' is displayed with 'v1' and 'OAS3' badges, and a link to the YAML file: '/api-docs/v1/swagger.yaml'. A 'Servers' section shows a dropdown with the URL 'https://{{defaultHost}}'. The computed URL is shown as 'Computed URL: https://stdc-api.herokuapp.com'. Under 'Server variables', there's a 'defaultHost' dropdown set to 'stdc-api.herokuapp.cc' and an 'Authorize' button with a lock icon. The main content area is divided into sections: 'Health' (with a GET /status endpoint), 'Attendances' (with five endpoints: GET /v1/attendances, POST /v1/attendances, GET /v1/attendances/{id}, PUT /v1/attendances/{id}, and GET /v1/attendances/all), and 'Course' (which is currently collapsed). Each endpoint includes a method, path, and a brief description.

API V1

v1 OAS3

/api-docs/v1/swagger.yaml

Servers

https://{{defaultHost}}

Computed URL: https://stdc-api.herokuapp.com

Server variables

defaultHost stdc-api.herokuapp.cc Authorize

Health

GET /status Retrieves the status of the application

Attendances

GET /v1/attendances Retrieves all attendances

POST /v1/attendances Creates a attendance

GET /v1/attendances/{id} Retrieves a attendance

PUT /v1/attendances/{id} Updates a attendance

GET /v1/attendances/all Retrieves all attendances (unpaginated)

Course

Figure 5.33: Swagger - Part 1

GET	/v1/courses	Retrieves all courses	▼	🔒
POST	/v1/courses	Creates a course	▼	🔒
GET	/v1/courses/{id}	Retrieves a course	▼	🔒
PUT	/v1/courses/{id}	Updates a course	▼	🔒
GET	/v1/courses/all	Retrieves all courses (unpaginated)	▼	🔒
GET	/v1/courses/{id}/students	Retrieves all students for a course	▼	🔒
POST	/v1/courses/assign-tutor	Assigns a tutor to a course	▼	🔒
Feedbacks				
GET	/v1/feedbacks	Retrieves all feedbacks	▼	🔒
POST	/v1/feedbacks	Creates a feedback	▼	🔒
GET	/v1/feedbacks/{id}	Retrieves a feedback	▼	🔒
PUT	/v1/feedbacks/{id}	Updates a feedback	▼	🔒
GET	/v1/feedbacks/all	Retrieves all feedbacks (unpaginated)	▼	🔒
Materials				
GET	/v1/materials	Retrieves all materials	▼	🔒
POST	/v1/materials	Creates a material	▼	🔒
GET	/v1/materials/{id}	Retrieves a material	▼	🔒
PUT	/v1/materials/{id}	Updates a material	▼	🔒
GET	/v1/materials/all	Retrieves all materials (unpaginated)	▼	🔒

Figure 5.34: Swagger - Part 2

Questions

GET	/v1/questions	Retrieves all questions	▼	🔒
POST	/v1/questions	Creates a question	▼	🔒
GET	/v1/questions/{id}	Retrieves a question	▼	🔒
PUT	/v1/questions/{id}	Updates a question	▼	🔒
GET	/v1/questions/all	Retrieves all questions (unpaginated)	▼	🔒

Session

GET	/v1/sessions	Retrieves all sessions	▼	🔒
POST	/v1/sessions	Creates a session	▼	🔒
GET	/v1/sessions/{id}	Retrieves a session	▼	🔒
PUT	/v1/sessions/{id}	Updates a session	▼	🔒
GET	/v1/sessions/all	Retrieves all sessions (unpaginated)	▼	🔒
GET	/v1/sessions/{id}/attendees	Retrieves all attendees of a session	▼	🔒
POST	/v1/sessions/{id}/toggle-registration	Registers attendees to a session	▼	🔒
POST	/v1/sessions/{id}/enter-attendance	Registers attendance of attendees to a session	▼	🔒
POST	/v1/sessions/{id}/remove-attendance	Removes attendance of attendees to a session	▼	🔒

User

GET	/v1/users	Retrieves all users	▼	🔒
POST	/v1/users	Creates a user	▼	🔒

Figure 5.35: Swagger - Part 3

The screenshot shows the Swagger UI interface for a RESTful API. It is organized into two main sections: 'User' and 'Venue'. Each section contains several API endpoints listed in a table-like format.

User Section:

Method	Path	Description	Lock icon
GET	/v1/users/{id}	Retrieves a user	🔓
PUT	/v1/users/{id}	Updates a user	🔒
GET	/v1/users/all	Retrieves all users (unpaginated)	🔓
GET	/v1/current-user	Retrieves the current user	🔓
POST	/v1/users/{id}/enter-availability	Enters a user's availability	🔒

Venue Section:

Method	Path	Description	Lock icon
GET	/v1/venues	Retrieves all venues	🔓
POST	/v1/venues	Creates a venue	🔓
GET	/v1/venues/{id}	Retrieves a venue	🔓
PUT	/v1/venues/{id}	Updates a venue	🔒
GET	/v1/venues/all	Retrieves all venues (unpaginated)	🔓

Figure 5.36: Swagger - Part 4

POST /v1/courses Creates a course

Parameters

Name	Description
Accept-Language string (headers)	Preferred language for the response Accept-Language

Request body

application/json

Example Value | Schema

```
{
  "data": {
    "status": "active",
    "name": "string",
    "userId": 0,
    "tutorId": 0
  }
}
```

Responses

Code	Description	Links
201	Success	No links

Media type

application/json

Controls Accept header

Example Value | Schema

```
{
  "data": {
    "id": "string",
    "type": "course",
    "attributes": {
      "id": 0,
      "status": "active",
      "name": "string",
      "rating": 0,
      "tutor": "string",
      "tutorId": "string",
      "nextSession": "string",
      "nextSessionVenue": "string",
      "createdAt": "string",
      "updatedAt": "string"
    },
    "jsonapi": {
      "version": "1.0"
    }
  }
}
```

Figure 5.37: Swagger - Course Post's Request and Response

Summary

In addition to the discussed aspects, there are several areas in the backend of the STDC web application that can be improved to achieve optimal performance. However, due to time constraints, only a limited number of improvements have been implemented. Some of the areas that require attention include resolving the tutor's association with a session and a course, addressing potential errors and suboptimal coding practices, and enhancing code quality.

Considering the context of the project, it is worth exploring the concept of microservices for the STDC application. As the university is relatively large, implementing a microservice architecture could offer significant advantages. This approach involves breaking down the application into separate services, such as user management and notifications. Each department could have its own system, integrated with others, resulting in a more scalable and modular application architecture.

Furthermore, it is important to acknowledge that while the backend meets the functional requirements, the code quality can be further improved. Given the time constraints and limitations, the current implementation has been a commendable effort. However, to ensure long-term maintainability and scalability, it is necessary to address code clean-ups and implement various improvements.

In conclusion, the backend implementation of the STDC web application using Ruby on Rails has provided a solid foundation for the overall functionality of the system. The deployment on Heroku ensures accessibility, and the provision of API endpoints facilitates seamless communication with the frontend. While only the important aspects have been covered in this section, there are numerous other considerations and enhancements required for a comprehensive and refined application.

To achieve the optimal state, future work should focus on resolving the tutor association, improving code quality, and exploring microservice architecture possibilities. With further development and refinement, the backend of the STDC web application can deliver an even more robust and efficient user experience.

5.1.2 Frontend Implementation (Client)

The following section shows how the implementation for the Client (the frontend) has been done for the Student Talent Development Center Web Application.

Technologies Used

The frontend of the application for the Student Talent Development Center (STDC) makes use of a variety of technologies in order to create a user interface that is both straightforward and interesting to interact with.

The frontend is built on top of a library written in JavaScript called React, which is used extensively. React is a framework that enables the rapid development of user interfaces for single-page or mobile applications. It was created and is maintained by Facebook in collaboration with a community of developers.

Auth0 is an authentication and authorization platform that makes user authentication more straightforward by supplying a solution that is friendly to developers. The application can effortlessly manage authentication procedures for its users as a result of its integration of Auth0. Auth0 has also been used in the backend as previously mentioned.

For React applications, React Query fills the role of the library that was previously lacking for fetching data. It simplifies the process of retrieving data from the server, as well as caching, synchronizing, and updating that data, which in turn improves the application's performance and efficiency.

Declarative composition is made possible within the application by utilizing the React Router, which is a collection of navigational components. It doesn't matter whether you're using it to generate bookmarkable URLs for web apps or to make navigation easier in React Native; React Router integrates with React in a seamless manner.

The use of React Hook Form allows for efficient management of the data collected from forms. This lightweight library, which does not depend on any external depen-

dencies, simplifies form data management. As a result, complexity is reduced, and development efficiency is increased.

A simple and straightforward approach to incorporating notifications into the application is provided by the React Toastify library. It provides a user-friendly option for displaying important messages to the users of the system.

Google's Material Design was adapted into an implementation called Material UI, which is a comprehensive collection of React components. The frontend of the STDC application adheres to established design principles and ensures a visually appealing and consistent user interface by utilizing Material UI. These goals were accomplished successfully.

A well-known promise-based HTTP client called Axios makes it simple to exchange information between the frontend and the backend of a website. It provides a user-friendly application programming interface (API) for submitting HTTP requests in both the browser and Node.js environments, in our case, React JS.

Redux, which is a container that provides a predictable state, is an essential component in the process of managing the application's state. It makes it possible to have consistent behavior across a variety of environments, such as the client, the server, and native platforms, while also improving the testability of the system.

A specialized library called the mui-rhf-library has been developed to integrate Material UI and React Hook Form. It makes the process of integration easier and ensures that there will be no hiccups in the interaction between the two libraries.

When it comes to describing, producing, consuming, and visually representing RESTful web services, OpenAPI, which is a specification for machine-readable interface files, is an extremely important component. It is put to use to generate services that are based on the API, which promotes the efficient and standardized development of software.

Yup is a JavaScript schema builder that makes it possible to validate and parse data

values. It provides a reliable solution for validating the data within the application, helping to guarantee the accuracy of the data as well as its integrity.

Because it uses a combination of these frontend technologies, the STDC application is able to provide users with an experience that is seamless, interactive, and visually appealing. Additionally, the application is able to manage data fetching, form handling, authentication, and state management in an efficient manner.

Authentication

Auth0 allows you to add authentication to your React application quickly and to gain access to user profile information. It also enables you to use a variety of authentication methods, including social media, username and password, and multi-factor authentication. The authentication process is handled entirely by Auth0, which means that you do not have to worry about implementing authentication protocols. The backend of the STDC application has been configured to use Auth0 for authentication purposes. Now, let's take a look at how the Auth0 has been integrated into the frontend of the STDC application.

First of all, in order to use Auth0 in the frontend, you need to install the Auth0 React SDK. The Auth0 React SDK is a library that allows you to integrate Auth0 into your React application. The Auth0 integration also requires an Auth0 application to be made. The process of how to create an Auth0 application is not mentioned here since it is out of the scope of this section. Nevertheless some of the configuration that need to be done in the process are crucial to mention here.

- **Application Keys:** In order to communicate with Auth0, the frontend needs to have the application keys. The application keys are the client ID and the client secret. The client ID and the client secret are used to identify the application to Auth0. The client ID and the client secret are used to generate the access token and the refresh token. The access token and the refresh token are used to authenticate the user.

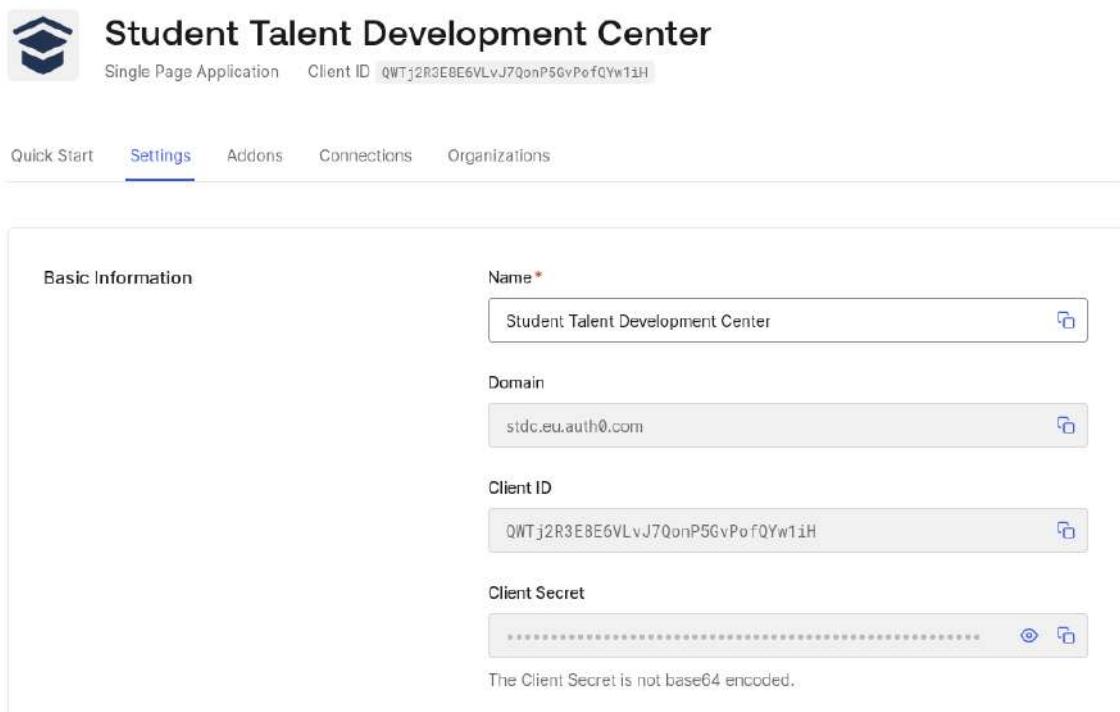


Figure 5.38: Auth0 Dashboard - Application Keys

- Allowed Callback URLs: The allowed callback URLs are the URLs that Auth0 will redirect the user to after the user has been authenticated. The allowed callback URLs are used to redirect the user to the frontend after the user has been authenticated. It is configured in the Auth0 application using the Auth0 dashboard.
- Allowed Logout URLs: The allowed logout URLs are the URLs that Auth0 will redirect the user to after the user has been logged out. The allowed logout URLs are used to redirect the user to the frontend after the user has been logged out. It is configured in the Auth0 application using the Auth0 dashboard.
- Allowed Web Origins: The allowed web origins are the URLs that Auth0 will use to silently authenticate the user. The allowed web origins are used to silently authenticate the user. It is configured in the Auth0 application using the Auth0 dashboard.

Allowed Callback URLs

```
https://oauth.pstmn.io/v1/callback, http://localhost:3000, https://stdc-api.herokuapp.com/, https://main--stdc.netlify.app/, https://dev--stdc.netlify.app/, https://stdc.netlify.app/, http://localhost:3000?*,
```

After the user authenticates we will only call back to any of these URLs. You can specify multiple valid URLs by comma-separating them (typically to handle different environments like QA or testing). Make sure to specify the protocol (`https://`) otherwise the callback may fail in some cases. With the exception of custom URI schemes for native clients, all callbacks should use protocol `https://`. You can use [Organization URL](#) parameters in these URLs.

Allowed Logout URLs

```
http://localhost:3000?*, https://stdc-api.herokuapp.com?*, https://main--stdc.netlify.app?*, https://stdc.netlify.app?*, https://stdc.onrender.com?*
```

A set of URLs that are valid to redirect to after logout from Auth0. After a user logs out from Auth0 you can redirect them with the `returnTo` query parameter. The URL that you use in `returnTo` must be listed here. You can specify multiple valid URLs by comma-separating them. You can use the star symbol as a wildcard for subdomains (`*.google.com`). Query strings and hash information are not taken into account when validating these URLs. Read more about this at [https://auth0.com/docs/authenticate/login/logout](#)

Allowed Web Origins

```
http://localhost:3000, https://stdc-api.herokuapp.com/, https://main--stdc.netlify.app/, https://dev--stdc.netlify.app/, https://stdc.netlify.app/, https://stdc-api.herokuapp.com, https://stdc.onrender.com/,
```

Comma-separated list of allowed origins for use with [Cross-Origin Authentication](#), [Device Flow](#), and [web message response mode](#), in the form of `<scheme> ":" <host> [":" <port>]`, such as `https://login.mydomain.com` or `http://localhost:3000`. You can use wildcards at the subdomain level (e.g.: `https://*.contoso.com`). Query strings and hash information are not taken into account when validating these URLs.

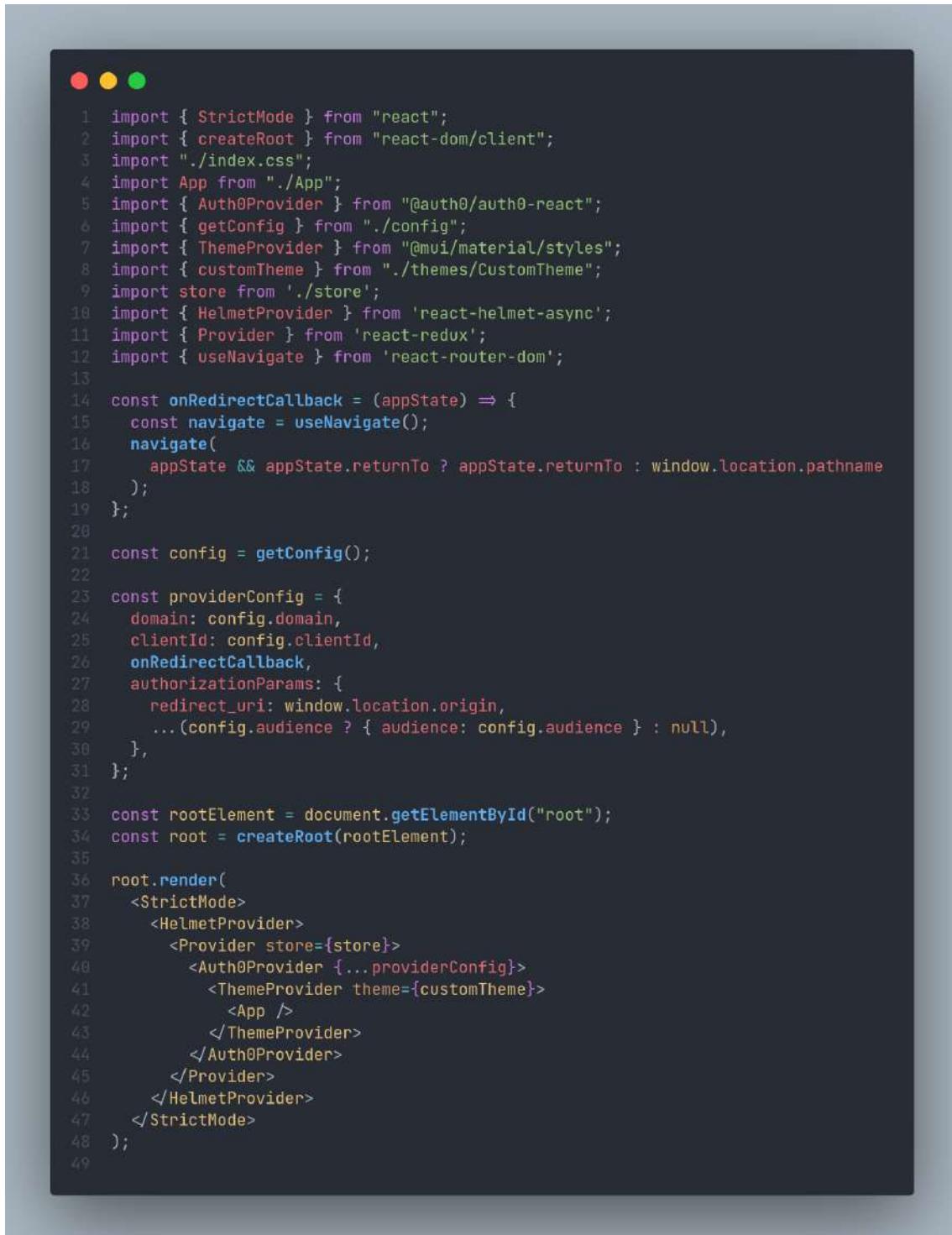
Figure 5.39: Auth0 Dashboard - Allowed URLs

In order to use Auth0, the `AuthProvider` component needs to be wrapped around the application. The `AuthProvider` component is responsible for initializing the Auth0

SDK and providing the Auth0 SDK to the application. The Auth0Provider component is defined as follows:

The Auth0Provider component is wrapped around the application in the index.js file.

The index.js file is the entry point of the application. The index.js file is as follows:

A screenshot of a code editor window showing the contents of the index.js file. The code is written in JavaScript and uses syntax highlighting. It imports various components and providers from external packages like react, react-dom/client, @auth0/auth0-react, and @mui/material/styles. The code defines a function onRedirectCallback that handles navigation based on app state. It also configures an Auth0Provider with providerConfig, which includes domain, clientId, and onRedirectCallback. The providerConfig also specifies authorizationParams with redirect_uri and audience. Finally, the root element is created and rendered with StrictMode, HelmetProvider, Provider, Auth0Provider, ThemeProvider, and App components.

```
1 import { StrictMode } from "react";
2 import { createRoot } from "react-dom/client";
3 import "./index.css";
4 import App from "./App";
5 import { Auth0Provider } from "@auth0/auth0-react";
6 import { getConfig } from "./config";
7 import { ThemeProvider } from "@mui/material/styles";
8 import { customTheme } from "./themes/CustomTheme";
9 import store from './store';
10 import { HelmetProvider } from 'react-helmet-async';
11 import { Provider } from 'react-redux';
12 import { useNavigate } from 'react-router-dom';
13
14 const onRedirectCallback = (appState) => {
15   const navigate = useNavigate();
16   navigate(
17     appState && appState.returnTo ? appState.returnTo : window.location.pathname
18   );
19 };
20
21 const config = getConfig();
22
23 const providerConfig = {
24   domain: config.domain,
25   clientId: config.clientId,
26   onRedirectCallback,
27   authorizationParams: {
28     redirect_uri: window.location.origin,
29     ...(config.audience ? { audience: config.audience } : null),
30   },
31 };
32
33 const rootElement = document.getElementById("root");
34 const root = createRoot(rootElement);
35
36 root.render(
37   <StrictMode>
38     <HelmetProvider>
39       <Provider store={store}>
40         <Auth0Provider {...providerConfig}>
41           <ThemeProvider theme={customTheme}>
42             <App />
43           </ThemeProvider>
44         </Auth0Provider>
45       </Provider>
46     </HelmetProvider>
47   </StrictMode>
48 );
49
```

Figure 5.40: Index.js

The Auth0Provider wrapper has several properties that need to be configured. The properties are domain, clientID, redirectURI, audience, and onRedirectCallback. The domain property is the domain of the Auth0 application. The clientID property is the client ID of the Auth0 application. The redirectURI property is the URL that Auth0 will redirect the user to after the user has been authenticated. The audience property is the audience of the Auth0 application. The onRedirectCallback property is the callback function that will be called after the user has been authenticated. All of these properties are passed down as a prop using providerConfig constant.

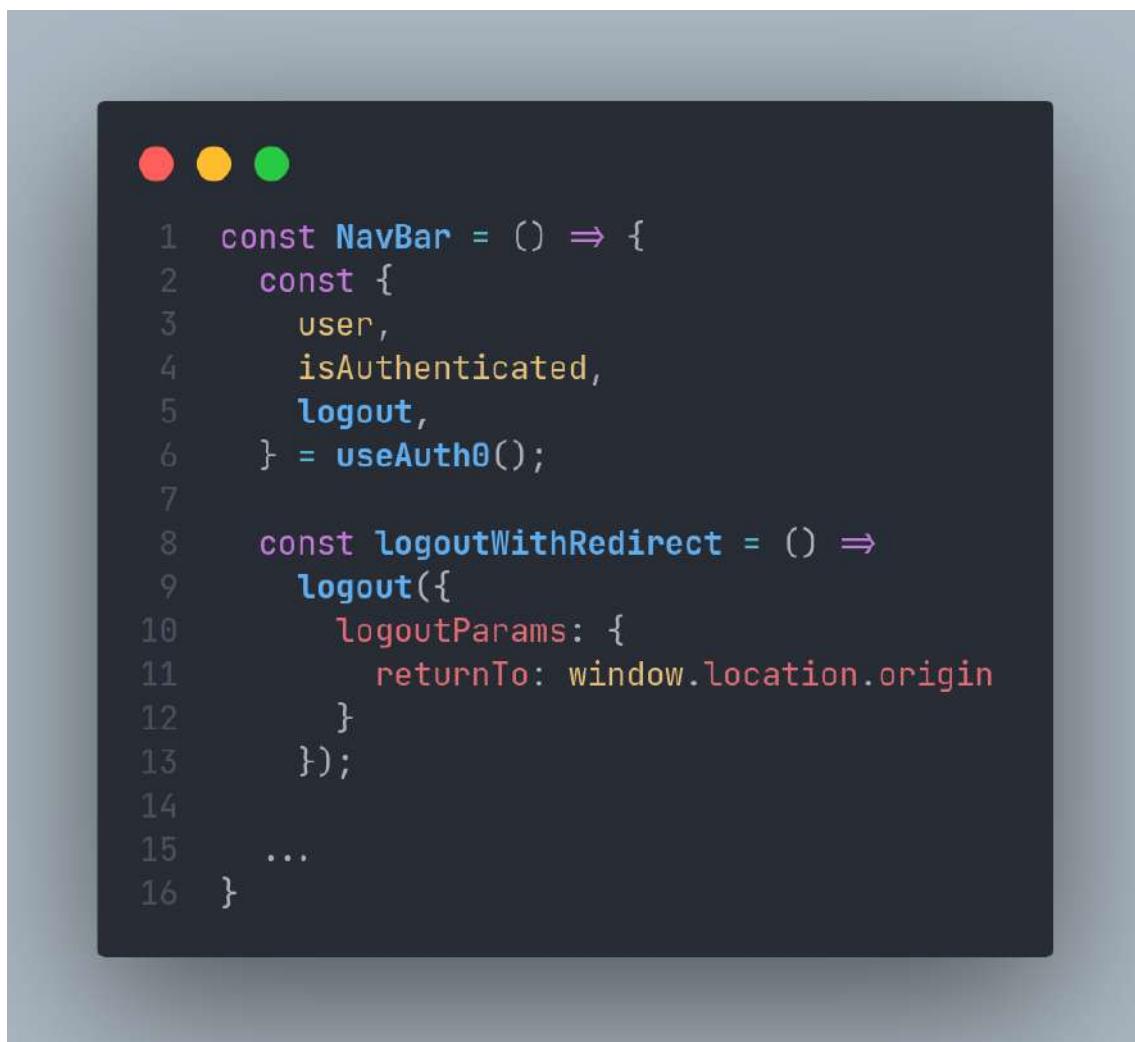
Now that the setup is done, let's take a look at logging in. The login process is handled by the LandingPage component. The LandingPage component is responsible for displaying the login page and handling the login process. The LandingPage component has a button that when clicked, it will call the loginWithRedirect method. The loginWithRedirect method is a method provided by the Auth0 SDK. The loginWithRedirect method is responsible for redirecting the user to the Auth0 login page. Once the user is redirected to the Auth0 login page, the user will be asked to enter their credentials. After the user has entered their credentials, Auth0 will redirect the user to the frontend. The frontend will then call the onRedirectCallback method. The onRedirectCallback method is a method provided by the Auth0 SDK. The onRedirectCallback method is responsible for handling the authentication process. The onRedirectCallback method redirects the user to the homepage if the user has been authenticated successfully. The onRedirectCallback method redirects the user to the login page if the user has not been authenticated successfully.

```
1 const LandingPage = () => {
2     const {
3         isAuthenticated,
4         loginWithRedirect
5     } = useAuth0();
6
7     return (
8         <>
9             {!isAuthenticated && (
10                 <>
11                     </>
12                 )
13             }
14         </>
15     )
16 };
17
18 export default LandingPage;
```

Figure 5.41: Landing Page - Login Code Block

The useAuth0 hook is a custom hook that is used to access the Auth0 SDK. The useAuth0 hook offers a variety of methods that can be used to interact with the Auth0 SDK including those mentioned above. Another useful property is isAuthenticated, which is a boolean value that indicates whether or not the user is authenticated. The isAuthenticated property is used to determine whether or not the user is authenticated. If the user is authenticated, the user will be redirected to the homepage. If the user is not authenticated, the user will be redirected to the login page.

Next, let's have a look at the logout process. The logout process is handled by the NavBar component. The NavBar component is responsible for displaying the navigation bar and handling the logout process. The NavBar component has a user profile icon that when clicked, it will show a menu of option, including visiting profile and logging out. When logout is clicked, it will call the logoutWithRedirect method. The logoutWithRedirect method is a method provided by the Auth0 SDK. The logoutWithRedirect method is responsible for redirecting the user to the Auth0 logout page. After the user has logged out, Auth0 will redirect the user to the frontend. The frontend will then redirects the user to the login page if the user has been logged out successfully.



```
1  const NavBar = () => {
2    const {
3      user,
4      isAuthenticated,
5      logout,
6    } = useAuth0();
7
8    const logoutWithRedirect = () =>
9      logout({
10        logoutParams: {
11          returnTo: window.location.origin
12        }
13      });
14
15    ...
16  }
```

Figure 5.42: NavBar - Logout Code Block

The Auth0 setup and configurations, login, and logout have been discussed in this section. Before Authorization is discussed, it is essential to discuss and understand how a logged in user is keeping their sessions. The process of setting up a current user is discussed below here.

The App component is the entry point of the application. The App component is responsible for setting up the application and rendering the application. The App component is where the set up of the currentUser is done. First, the StdcApiOpenAPI is configured, more on this in the coming sections. The StdcApiOpenAPI is used to communicate with the backend. Then, the Auth0 SDK is used to get the access token silently. The access token is used to authenticate the user. The access token is used to authenticate the user by passing it to the backend. The backend will then verify the access token and return the user if the access token is valid. The access token is stored in the local storage to be used later with every request to the backend. Since the current user is set, the application can now be rendered. The current user can be accessed anywhere as long as they are authenticated.

```
1 function App() {
2   StdcApiOpenAPI.BASE = 'https://stdc-api.herokuapp.com';
3   const queryClient = new QueryClient();
4
5   const { isLoading, getAccessTokenSilently } = useAuth0();
6   const [token, setToken] = React.useState(null);
7   const dispatch = useDispatch();
8
9   const getToken = async () => {
10     const token = await getAccessTokenSilently();
11     setToken(token);
12   };
13
14   getToken();
15
16   useEffect(() => {
17     if (token) {
18       setHttpToken(token);
19       localStorage.setItem('token', token);
20     }
21   }, [token, setHttpToken]);
22
23   const fetchUser = useCallback(async () => {
24     const res = await fetch('https://stdc-api.herokuapp.com/v1/current-user', {
25       headers: {
26         Authorization: `Bearer ${token}`,
27       },
28     });
29     const user = await res.json();
30
31     dispatch(setUser(user));
32     dispatch(setIsAuthenticatingLoading(false));
33   }, [token, dispatch]);
34
35   useEffect(() => {
36     if (token) {
37       fetchUser();
38     }
39   }, [token, fetchUser]);
40
41   if (isLoading) {
42     return <Loading />;
43   }
44
45   ...
46 }
47
```

Figure 5.43: App.js

As previously mentioned, the backend will be handling authentication for the most part. As for the client side, obtaining the token and keeping the user in a session was the objective. There are more details when it comes to authentication, but the mentioned parts are the most important parts of the authentication process. The

other parts of the authentication process are not mentioned in this section because they can be abstracted away. In brief, now we know how authentication works. Next let's take a look onto authorization.

Authorizations

The Student Talent Development Center (STDC)’s client (frontend) routes are managed by the react-router-dom library. The react router library provides a BrowserRouter component that wraps the entire application. The BrowserRouter component is responsible for managing the application’s routes and rendering the appropriate components based on the current route. Usually React Router DOM requires set of routes to be defined in a tree structure, where each route can have a parent route, and a list of child routes. The App.js file is the entry point of the application after Index.js, and it is where the BrowserRouter component is defined. In order to separate the concerns, the STDC has its route defined in another place. Then it imports them to the RouterProvider component, and passes them to the BrowserRouter component.



```
1  function App() {
2
3    ...
4
5    const routing = createBrowserRouter(routes);
6
7    return (
8      <>
9        <Toaster ... />
10       <QueryClientProvider client={queryClient}>
11         <RouterProvider router={routing} />
12       </QueryClientProvider >
13     </>
14   );
15 }
16
17 export default App;
18
```

Figure 5.44: Routes - App.js

The routes are defined in the routes.js file. The routes are defined in a tree structure, where each route can have a parent route, and a list of child routes. There are two types of routes in the STDC client, protected routes, and public routes. The public routes are the routes that do not require the user to be authenticated to access them. Since STDC is by nature a protected application, the public routes are only the login and the static pages like the 404 page. The protected routes are the routes that require the user to be authenticated to access them. The protected routes are defined in the protectedRoutes array. Despite the fact that the STDC requires authentication to access any of its routes, it also requires authorization to access some of its routes. Each route has a property called "roles", which is an array of strings that represent the roles that are allowed to access the route. The roles are defined in the currentUser object, which is a part of the user slice in the Redux store more on this later. When a user tries to access a route, first the user's roles are checked against the roles of the route, if the user's roles include one of the roles of the route, then the user is authorized to access the route. If the user is not authorized to access the route, then the user is redirected to 403 Forbidden page. In case that the user is not authenticated, then the user is redirected to 401 Unauthorized page. If the route does not exist, then the user is redirected to 404 Not Found page. All of these static pages, 401, 403, and 404 are public, and offer the user to navigate to the home page, or to login page if not authenticated.

```
1 const routes = [
2   {
3     path: '*',
4     element: <BaseLayout />,
5
6     children: [
7       {
8         path: '*',
9         element: <Status404 />
10      },
11      {
12        path: 'login',
13        element: (
14          <Login />
15        )
16      },
17      {
18        path: '403',
19        element: <Status403 />
20      }
21    ],
22  },
23  ...
24 ]
25 ]
26 ]
```

Figure 5.45: Public Routes - routes.js

```
1 const routes = [
2 ...
3 ...
4 {
5     path: '*',
6     element: (
7         <ProtectedRoute
8             roles={['superadmin', 'viewer', 'student', 'tutor']}
9             ...
10            >
11                <SidebarLayout />
12            </ProtectedRoute>
13        ),
14        children: [
15            {
16                index: true,
17                element: (
18                    <ProtectedRoute
19                        roles={['superadmin', 'viewer', 'student', 'tutor']}
20                        ...
21                        >
22                            <Navigate to="/home" replace />
23                        </ProtectedRoute>
24                )
25            },
26            {
27                path: 'home',
28                element: (
29                    <ProtectedRoute
30                        roles={['superadmin', 'viewer', 'student', 'tutor']}
31                        routeWhenNoAccess={<Navigate to={'/profile'} />}
32                        ...
33                        >
34                            <Home />
35                        </ProtectedRoute>
36                )
37            },
38            ...
39        ],
40        ...
41        {
42            path: 'courses',
43            element: (
44                <ProtectedRoute
45                    roles={['superadmin', 'viewer', 'student', 'tutor']}
46                    routeWhenNoAccess={<Navigate to={'/profile'} />}
47                    ...
48                    >
49                        <Course />
50                    </ProtectedRoute>
51                )
52            },
53            ...
54        ],
55    },
56];
57
58 export default routes;
```

Figure 5.46: Protected Routes - routes.js



```
const ProtectedRoute = ({ roles, children, routeWhenNoAccess }) => {
  const { isAuthenticated, isAuthenticationLoading, currentUser } = useSelector((state) => state.user);
  if (isAuthenticationLoading) return <Login />

  if (roles) {
    return isAuthenticated ? (
      currentUser?.data?.attributes?.roles?.includes('superadmin') || roles.find((role) => currentUser?.data?.attributes?.roles?.includes(role)) ? (
        children
      ) : (
        routeWhenNoAccess || <Navigate to={'/403'} />
      )
    ) : (
      <Navigate to={'/login'} />
    );
  } else {
    return isAuthenticated ? children : <Navigate to={'/login'} />;
  }
};
```

Figure 5.47: Protected Routes - hasAccessTo Usage

In the routes file, there is a function called `hasAccessTo`, which is responsible for checking if the user is authorized to access a route. The `hasAccessTo` function takes an object as an argument, and the object can have two properties, `permission`, and `roles`. The `permission` property is a string that represents the permission that the user needs to access the route. The `roles` property is an array of strings that represents the roles that are allowed to access the route. The `hasAccessTo` function checks if the user's roles include the permission, or one of the roles of the route. If the user's roles include the permission, or one of the roles of the route, then the user is authorized to access the route. If the user's roles do not include the permission, or one of the roles of the route, then the user is not authorized to access the route. The `hasAccessTo` function is a utility function that is used in many places in the STDC client, and it is used to check if the user is authorized to access a route, or to access a component.

A screenshot of a terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) at the top left. The terminal shows a block of JavaScript code with line numbers from 1 to 21 on the left. The code defines a function named hasAccessTo that takes an object with permission, roles, and permissions keys. It first checks if permission is present. If not, it returns false. Then, it checks if roles is present. If not, it returns false. Next, it checks if roles includes permission. If so, it sets hasAccess to true. Finally, it checks if permissions is present. If so, it loops through each permission in permissions. For each permission, it calls the hasAccessTo function with the current permission and roles. If any call returns true, it sets hasAccess to true and breaks out of the loop. If no permission in permissions has access, it returns false. The code ends with a return hasAccess; statement.

```
1 export const hasAccessTo = ({ permission, roles, permissions }) => {
2     let hasAccess = false;
3
4     if (permission) {
5         if (!roles) return;
6
7         if (roles.includes(permission)) {
8             hasAccess = true;
9         }
10    } else if (permissions) {
11        for (const permission of permissions) {
12            if (hasAccessTo({ permission, roles })) {
13                hasAccess = true;
14                break;
15            }
16        }
17    }
18
19    return hasAccess;
20};
21
```

Figure 5.48: hasAccessTo

Below is an example of how hasAccessTo is used in the STDC client. The example is from the Sidebar component, which is the component that renders the sidebar of the application. In the Sidebar component, there is a list of clickable items, each item represents a route in the application. Depending on the user's roles, some of the items are hidden, and some of the items are shown. The below block of code, shows how the hasAccessTo function is used to check if the user is authorized to access the courses route. Only if the user's roles include the superadmin role, then the courses item is shown in the sidebar. Else, the courses item is hidden from the sidebar.



A screenshot of a code editor window showing a React component. The component uses the `hasAccessTo` function to filter a list of items based on user permissions and roles. The code includes imports for `AutoAwesomeMosaicIcon`, `ListItem`, `ListItemIcon`, `ListItemText`, `ListItemButton`, and `NavLink`. The component checks if the user has access to 'superadmin' permissions and then filters the `roles` array. It then maps over the filtered roles to create `NavLink` components, each with a `ListItemIcon` containing an `AutoAwesomeMosaicIcon` and a `ListItemText` with the primary text 'Courses'.

```
1 {hasAccessTo({
2     permissions: ['superadmin'],
3     roles: currentUser?.data?.attributes?.roles?.map((role) => role)
4 }) && (
5     <NavLink to="/courses" style={{ textDecoration: 'none' }}>
6         <ListItem key="courses" disablePadding>
7             <ListItemButton>
8                 <ListItemIcon>
9                     <AutoAwesomeMosaicIcon color="textDark" />
10                </ListItemIcon>
11                <ListItemText primary="Courses" />
12            </ListItemButton>
13        </ListItem>
14    </NavLink>
15 )
16 }
```

Figure 5.49: `hasAccessTo` - Sidebar Example

Another example of how `hasAccessTo` is used in the STDC client, is in the Show Session component. The Show Session component is the component that renders the a specific session's page. In the session page, there are a list of tabs, each tab represents a panel in the session page. Not all of the tabs are shown to the user. Depending on the user's roles, some of the tabs are hidden, and some of the tabs are shown same as the sidebar. The below block of code, shows how the `hasAccessTo` function is used to check if the user is authorized to access a tab.

```
1  {
2    (currentUser?.data?.attributes?.roles?.map((role) => role).includes('superadmin')) ? (
3      hasAccessTo({
4        permissions: ['superadmin'],
5        roles: currentUser?.data?.attributes?.roles?.map((role) => role)
6      }) && (
7        <>
8          <Tabs value={value} onChange={handleChange}
9            ...
10         >
11           <LinkTab label="Properties" {...a11yProps(0)} href="/properties" />
12           <LinkTab label="Attendance" {...a11yProps(1)} href={`/attendance`} />
13           <LinkTab label="Registered Students" {...a11yProps(2)} href="/registered-students" />
14           <LinkTab label="Assign Time and Venue" {...a11yProps(3)} href="/assign-time-and-venue" />
15           <LinkTab label="Provide Feedback" {...a11yProps(4)} href="/provide-feedback" />
16           <LinkTab label="Upload Material" {...a11yProps(5)} href="/upload-material" />
17           <LinkTab label="Materials" {...a11yProps(6)} href="/materials" />
18         </Tabs>
19       </>
20     )
21   ) : (currentUser?.data?.attributes?.roles?.map((role) => role).includes('tutor')) ? (
22     hasAccessTo({
23       permissions: ['tutor'],
24       roles: currentUser?.data?.attributes?.roles?.map((role) => role)
25     }) && (
26       <>
27         <Tabs value={value} onChange={handleChange}
28           ...
29         >
30           <LinkTab label="Properties" {...a11yProps(0)} href="/properties" />
31           <LinkTab label="Attendance" {...a11yProps(1)} href={`/attendance`} />
32           <LinkTab label="Registered Students" {...a11yProps(2)} href="/registered-students" />
33           <LinkTab label="Upload Material" {...a11yProps(5)} href="/upload-material" />
34           <LinkTab label="Materials" {...a11yProps(6)} href="/materials" />
35         </Tabs>
36       </>

```

Figure 5.50: hasAccessTo - Panel Example Part 1



```
1      ) : (currentUser?.data?.attributes?.roles?.map((role) => role).includes('student')) ? (
2        hasAccessTo({
3          permissions: ['student'],
4          roles: currentUser?.data?.attributes?.roles?.map((role) => role)
5        }) && (
6          <>
7            <Tabs value={value} onChange={handleChange}>
8              ...
9              >
10             <LinkTab label="Properties" {...a11yProps(0)} href="/properties" />
11             <LinkTab label="Provide Feedback" {...a11yProps(4)} href="/provide-feedback" />
12             <LinkTab label="Materials" {...a11yProps(6)} href="/materials" />
13           </Tabs>
14         </>
15       )
16     )
17   ) : (
18     hasAccessTo({
19       permissions: ['viewer'],
20       roles: currentUser?.data?.attributes?.roles?.map((role) => role)
21     }) && (
22       <>
23         <Tabs value={value} onChange={handleChange}>
24           ...
25           >
26           <LinkTab label="Properties" {...a11yProps(0)} href="/properties" />
27           <LinkTab label="Materials" {...a11yProps(6)} href="/materials" />
28         </Tabs>
29       </>
30     )
31   )
32 )
33 }
```

Figure 5.51: hasAccessTo - Panel Example Part 2

Although the authorization that is done in the client side is mostly by hiding the components, and the routes, the STDC client also does through authorization in the server side. The server side authorization is done by checking the user's roles against the permissions of user which have been mentioned in the previous section 'Backend'. The client side also redirects users to the 401, 403, and 404 pages, if the user is not authorized to access a route, or a component.

In terms of security, the STDC client is a protected application, and it requires authentication and authorization to access any of its routes, and components. The way STDC deals with client side security might not be the best way, but it is a very good way to deal with client side security nonetheless.

API Communication

The Student Talent Development Center (STDC) client (frontend) communicates with the backend using the OpenAPI, Axios, Axios Interceptors, and React Query. The OpenAPI is used to generate services that are based on the API, which promotes the efficient and standardized development of software. Axios is a promise-based HTTP client that makes it simple to exchange information between the frontend and the backend of web applications. Interceptors are functions that can be used to intercept requests or responses before they are handled by the application. React Query is a library that makes it simple to fetch, cache, and update data in React applications. All of these technologies are integrated together to communicate with the backend. Now let's take a look at how the STDC client uses these technologies to communicate with the backend.

The OpenAPI is used to generate services that are based on the API, which is the swagger that the backend have generated. Each schema that the backend have made available in the backend will become available to client on the service generation. The generation of the services is done using the openapi-generator-cli package. The openapi-generator-cli package is a command line interface that can be used to generate services based on the API. The openapi-generator-cli package is configured in the package.json file. The package.json file is the file that contains the configuration of the STDC client.



```
1 "scripts": {
2   ...
3   "generate-stdc-api": "openapi -c axios --useOptions -i https://stdc-api.herokuapp.com/api-docs/v1/swagger.yaml -o src/services/api/stdc"
4 },
```

Figure 5.52: OpenAPI Command to Generate Services

In order for the above command to work, first the base address of the api needs to be set. This is done in the App.js file.



```
1 function App() {
2     StdcApiOpenAPI.BASE = 'https://stdc-api.herokuapp.com';
3     ...
4 }
```

Figure 5.53: OpenAPI Base Address

Once the command is set in the package.json file, and the base address is set in the App.js file, the services can be generated. The services are generated in the services folder. The services folder is the folder that contains the services that are generated based on the API. The services folder is located in the src folder. The src folder is the folder that contains the source code of the STDC client.

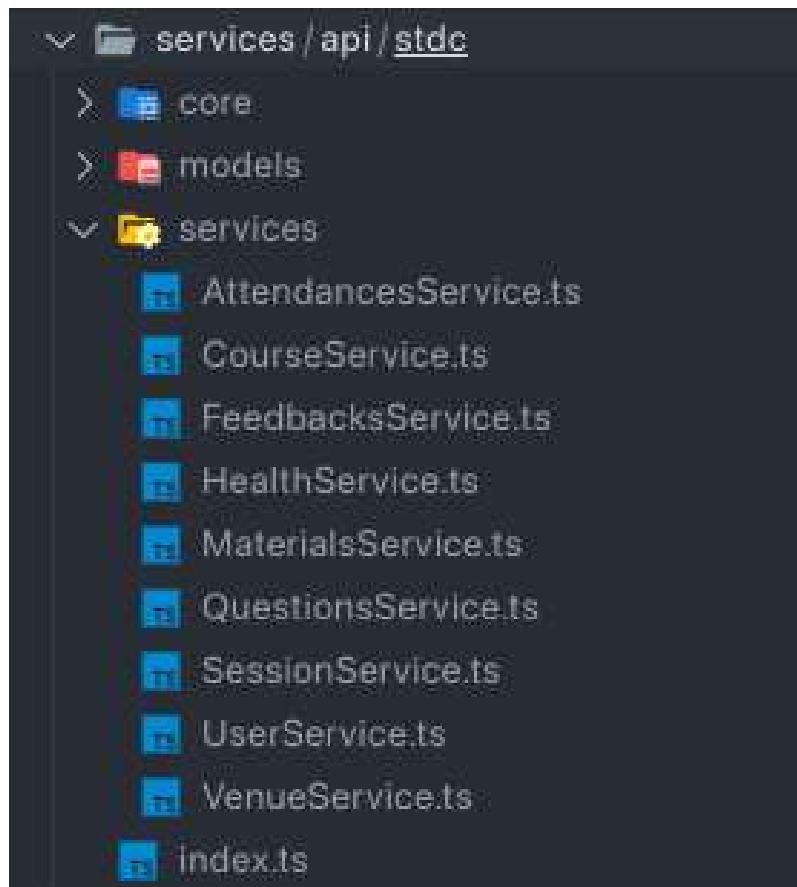


Figure 5.54: Services Folder

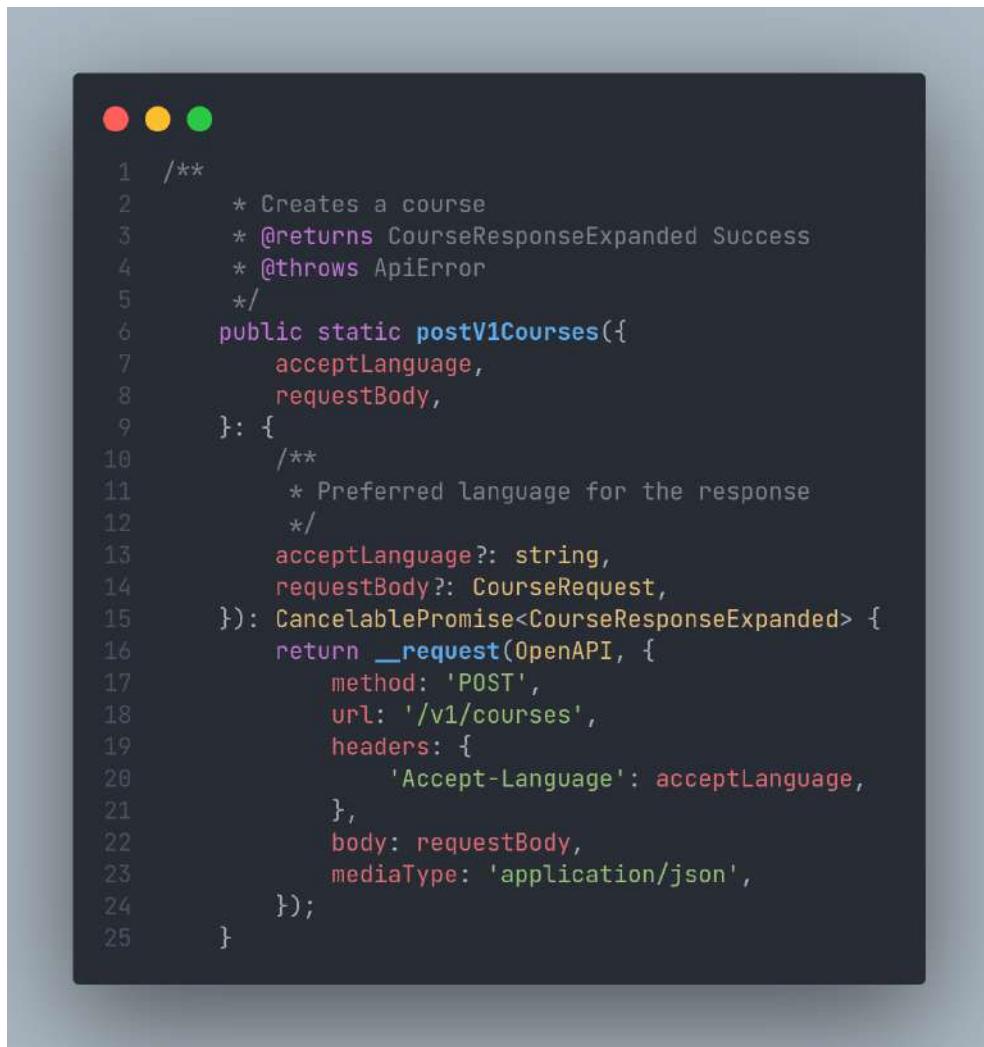
The following figure shows how the services are used to fetch courses from the backend.



```
1 const fetchAllCourses = async () => {
2     const res = await CourseService.getV1Courses({
3         status: hasAccessTo({
4             permissions: ['superadmin'],
5             roles: currentUser?.data?.attributes?.roles?.map((role) => role)
6         })
7         ? undefined
8         : 'active'
9     });
10    setAllCourses(res.data);
11};
```

Figure 5.55: Services Usage - Get Courses

The below service is generated by the Open API library. The service is used to allow for sending a post request of a course creation to the backend.



```
1 /**
2  * Creates a course
3  * @returns CourseResponseExpanded Success
4  * @throws ApiError
5  */
6 public static postV1Courses({
7     acceptLanguage,
8     requestBody,
9 }: {
10     /**
11      * Preferred language for the response
12      */
13     acceptLanguage?: string,
14     requestBody?: CourseRequest,
15 }): CancelablePromise<CourseResponseExpanded> {
16     return __request(OpenAPI, {
17         method: 'POST',
18         url: '/v1/courses',
19         headers: {
20             'Accept-Language': acceptLanguage,
21         },
22         body: requestBody,
23         mediaType: 'application/json',
24     });
25 }
```

Figure 5.56: Generated Service - Post Course Endpoint

React Query is another library that is used to communicate with the backend. React Query is a library that makes it simple to fetch, cache, and update data in React applications. React Query is configured in the App.js file as follows:

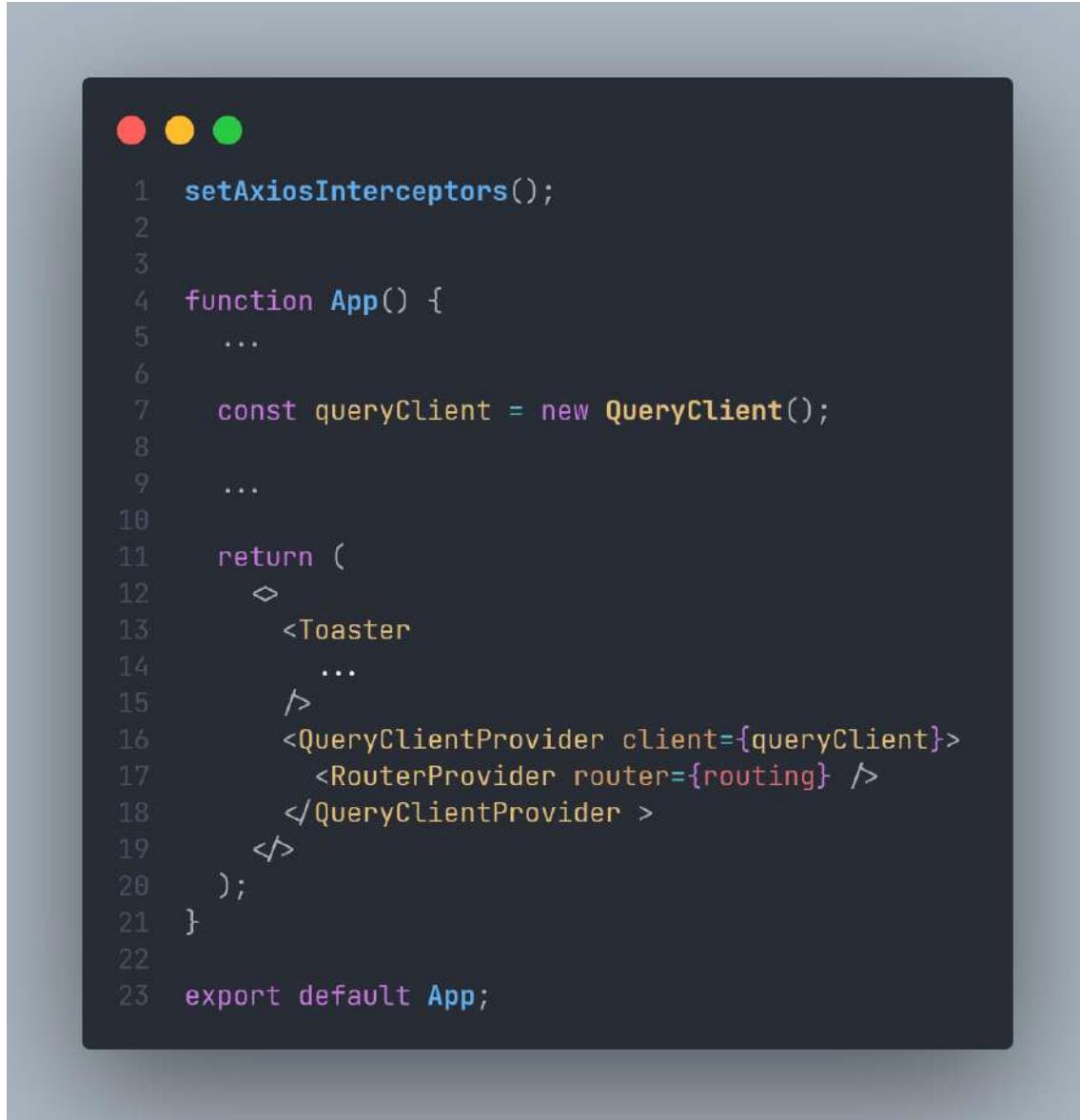
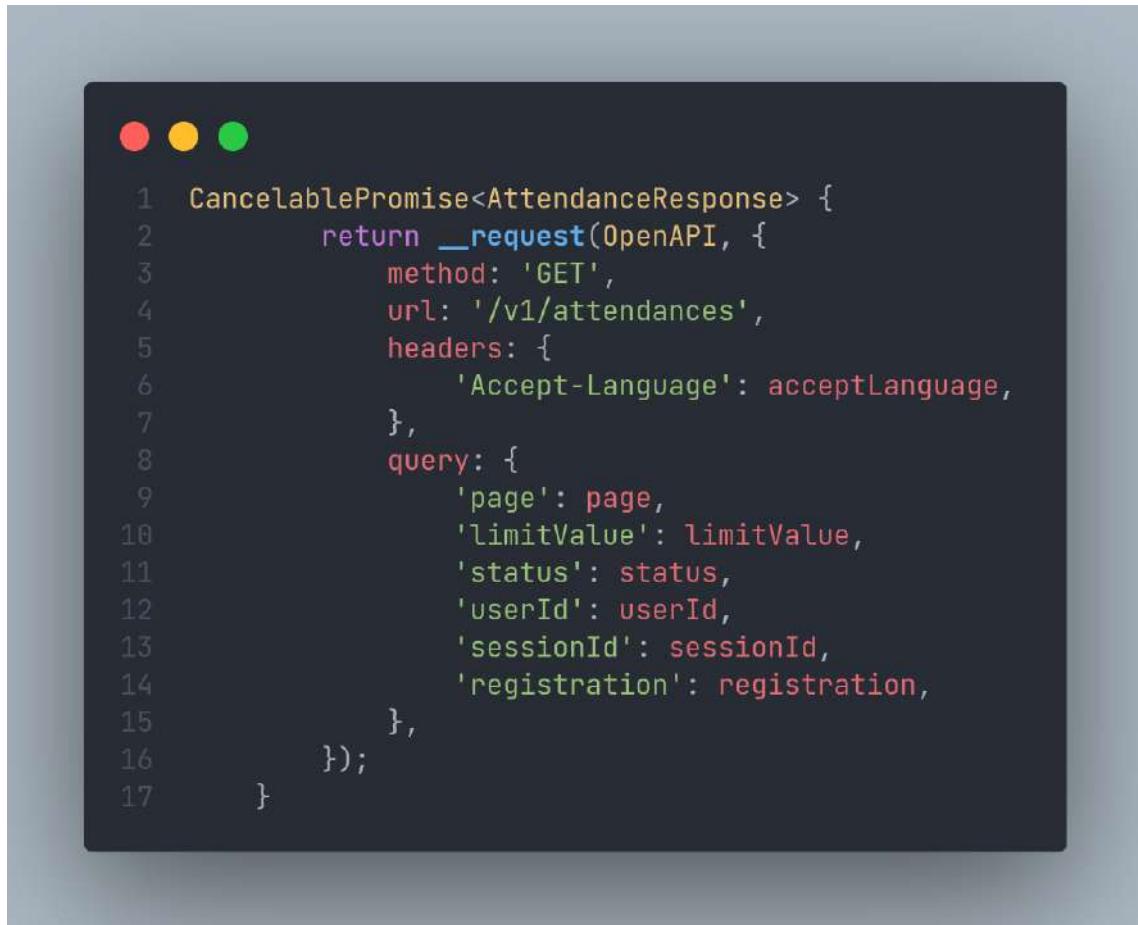
A screenshot of a Mac OS X desktop showing a terminal window. The window has three colored window control buttons (red, yellow, green) at the top. The terminal background is dark. The code is displayed in a light-colored font. The code is a JavaScript file named App.js, which defines a function App that returns a component tree starting with <Toaster>. It includes imports for setAxiosInterceptors, QueryClient, RouterProvider, and QueryClientProvider from react-query.

Figure 5.57: React Query Configuration

The component of the STDC have access to the React Query using the useQuery hook. The QueryClientProvider context provider is used to provide the React Query to the components. Component of the application use it the useQuery hook directly or through the services. The Services that are generated by the OpenAPI, makes use of React Query, the below example shows the services' usage of React Query.

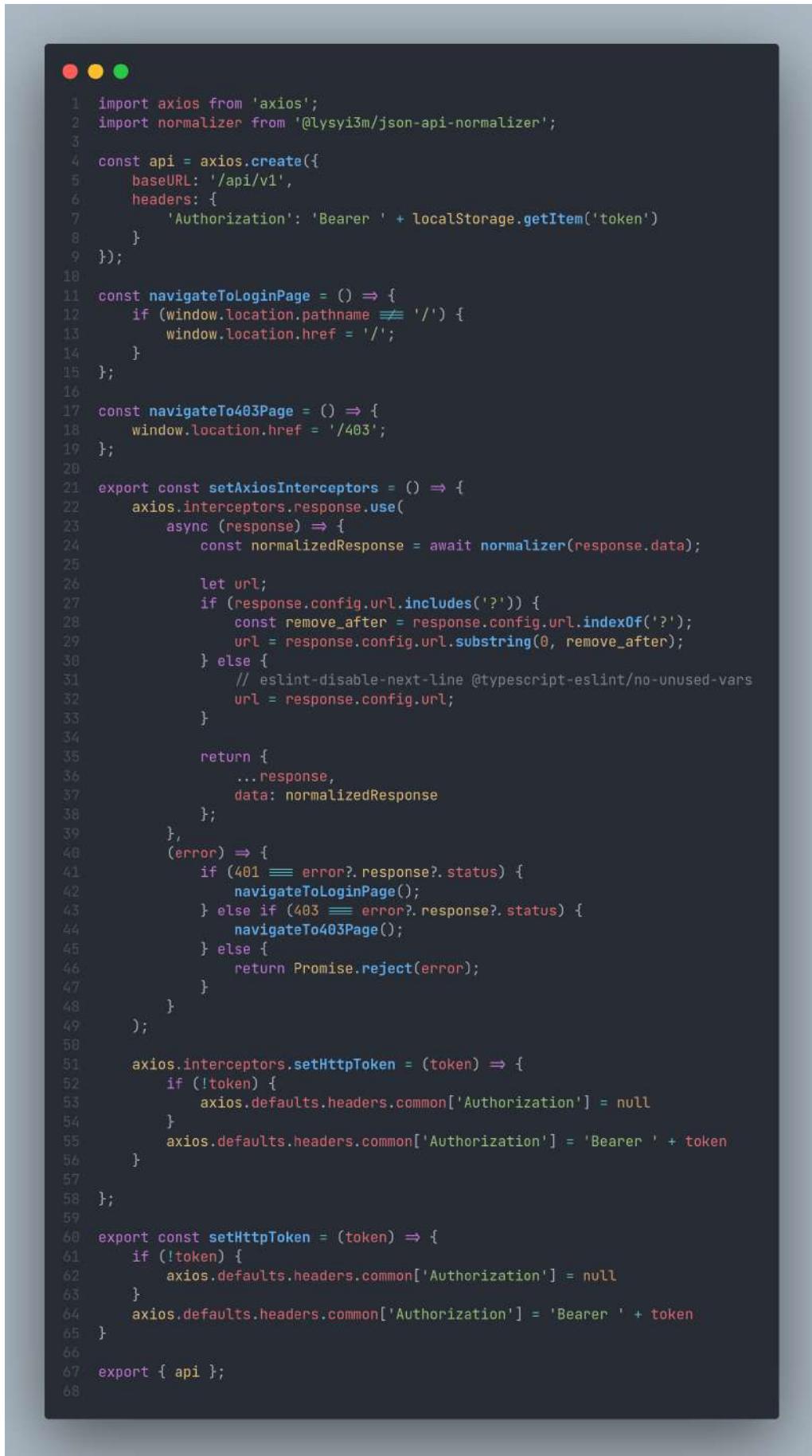


A screenshot of a terminal window with three colored window control buttons (red, yellow, green) at the top. The terminal displays a block of code in a monospaced font. The code is a JavaScript function named `getAttendances` that returns a `CancelablePromise` of type `AttendanceResponse`. It uses the `_request` method from the `OpenAPI` object, specifying a `GET` method, the URL `/v1/attendances`, and a header `'Accept-Language'` set to `acceptLanguage`. The `query` parameter is an object containing `page`, `limitValue`, `status`, `userId`, `sessionId`, and `registration`.

```
1 CancelablePromise<AttendanceResponse> {
2     return __request(OpenAPI, {
3         method: 'GET',
4         url: '/v1/attendances',
5         headers: {
6             'Accept-Language': acceptLanguage,
7         },
8         query: {
9             'page': page,
10            'limitValue': limitValue,
11            'status': status,
12            'userId': userId,
13            'sessionId': sessionId,
14            'registration': registration,
15        },
16    });
17}
```

Figure 5.58: Services' Query Usage

Axios is a promise-based HTTP client that makes it simple to exchange information between the frontend and the backend of web applications. Axios is configured inside the services and used by the OpenAPI to communicate with the backend. However, Axios has interceptors that can be used to intercept requests or responses before they are handled by the application. Interceptors are useful for a variety of reasons, including handling errors, adding headers to requests, and more. The figure 5.57 shows how the interceptors are configured in the `App.js` file. The below figure shows how the interceptors are configured in the STDC client.



```
1 import axios from 'axios';
2 import normalizer from '@lysyi3m/json-api-normalizer';
3
4 const api = axios.create({
5   baseURL: '/api/v1',
6   headers: {
7     'Authorization': 'Bearer ' + localStorage.getItem('token')
8   }
9 });
10
11 const navigateToLoginPage = () => {
12   if (window.location.pathname !== '/') {
13     window.location.href = '/';
14   }
15 };
16
17 const navigateTo403Page = () => {
18   window.location.href = '/403';
19 };
20
21 export const setAxiosInterceptors = () => {
22   axios.interceptors.response.use(
23     async (response) => {
24       const normalizedResponse = await normalizer(response.data);
25
26       let url;
27       if (response.config.url.includes('?')) {
28         const removeAfter = response.config.url.indexOf('?');
29         url = response.config.url.substring(0, removeAfter);
30       } else {
31         // eslint-disable-next-line @typescript-eslint/no-unused-vars
32         url = response.config.url;
33       }
34
35       return {
36         ...response,
37         data: normalizedResponse
38       };
39     },
40     (error) => {
41       if (401 === error?.response?.status) {
42         navigateToLoginPage();
43       } else if (403 === error?.response?.status) {
44         navigateTo403Page();
45       } else {
46         return Promise.reject(error);
47       }
48     }
49   );
50
51   axios.interceptors.setHttpToken = (token) => {
52     if (!token) {
53       axios.defaults.headers.common['Authorization'] = null
54     }
55     axios.defaults.headers.common['Authorization'] = 'Bearer ' + token
56   }
57
58 };
59
60 export const setHttpToken = (token) => {
61   if (!token) {
62     axios.defaults.headers.common['Authorization'] = null
63   }
64   axios.defaults.headers.common['Authorization'] = 'Bearer ' + token
65 }
66
67 export { api };
68
```

Figure 5.59: Axios Interceptors Configuration

The above figure shows that the interceptor for responses is used to intercept responses before they are handled by the application to normalize the responses before they are handled by the application. Another use of this interceptor is to redirect the user to the static pages of 401 and 403 if the user is not authenticated or not authorized to access the requested resource. The interceptor for requests is used to intercept requests before they are sent to the backend to add the access token to the request headers. The access token is used to authenticate the user by passing it to the backend. The backend will then verify the access token and return the requested resource upon authentication.

In this section, we talked about how the above technologies were integrated into the STDC client and how they have been utilized to communicate with the backend. Next, we will talk about how the STDC client handles its State Management.

State Management

In the process of developing React applications, state management is an essential component as the application requires the continuous synchronization of data across all of the different components. It is common practice to make use of the Redux library in React applications in order to manage the application's global state. Redux is a predictable state container. Redux makes it possible to organize and update state in a manner that is consistent and predictable by requiring that its users follow a predetermined set of steps. Explanation of how Redux work is not in the scope of this section. However, the STDC client uses Redux to manage its state. The STDC client uses Redux to manage its state in a manner that is consistent with the Redux's best practices. Now, let's take a look at how the STDC client uses Redux to manage its state.

The main structure of the STDC client's redux is as follows:

- The store is the object that holds the application's state.
- The reducer is a function that takes the current state and an action as arguments and returns the next state.
- The slice is a collection of reducer logic and actions for a single feature of the application's state.

The store is the object that holds the application's state. The store is as follows:



```
1 import { useDispatch as useReduxDispatch, useSelector as useReduxSelector } from 'react-redux';
2 import { configureStore } from '@reduxjs/toolkit';
3
4 import rootReducer from './rootReducer';
5
6 const store = configureStore({
7   reducer: rootReducer
8 });
9
10 export const useSelector = useReduxSelector;
11
12 export const useDispatch = () => useReduxDispatch();
13
14 export default store;
15
```

Figure 5.60: Store

STDC also has a rootReducer, which is a function that combines all of the reducers into a single reducer. The rootReducer is defined as follows:



```
 1 import { combineReducers } from '@reduxjs/toolkit';
 2 import { reducer as userReducer } from '../slices/user';
 3 import venueReducer from '../slices/venue';
 4 import courseReducer from '../slices/course';
 5 import feedbackReducer from '../slices/feedback';
 6 import sessionReducer from '../slices/session';
 7 import usersReducer from '../slices/users';
 8 import questionReducer from '../slices/question';
 9
10 const rootReducer = combineReducers({
11   user: userReducer,
12   venues: venueReducer,
13   courses: courseReducer,
14   users: usersReducer,
15   sessions: sessionReducer,
16   feedbacks: feedbackReducer,
17   questions: questionReducer
18 });
19
20 export default rootReducer;
21
```

Figure 5.61: Root Reducer

The STDC client's redux is divided into slices, each slice is responsible for managing a specific part of the application's state. The slices are as follows:

1. user: The user slice is responsible for managing the current user's state. The user's state includes the user's information.
2. venues: The venues slice is responsible for managing the venues' state. The venues' state includes the venues' information.
3. courses: The courses slice is responsible for managing the courses' state. The courses' state includes the courses' information.

4. users: The users slice is responsible for managing the users' state. The users' state includes the users' information.
5. sessions: The sessions slice is responsible for managing the sessions' state. The sessions' state includes the sessions' information.
6. feedbacks: The feedbacks slice is responsible for managing the feedbacks' state. The feedbacks' state includes the feedbacks' information.
7. questions: The questions slice is responsible for managing the questions' state. The questions' state includes the questions' information.

In general the STDC uses these slices to manage the state of the global filters of the resource tables.

The following is the code of the courses slice. The courses slice is responsible for managing the courses' state. The courses' state includes the courses' information. The courses slice is defined as follows.

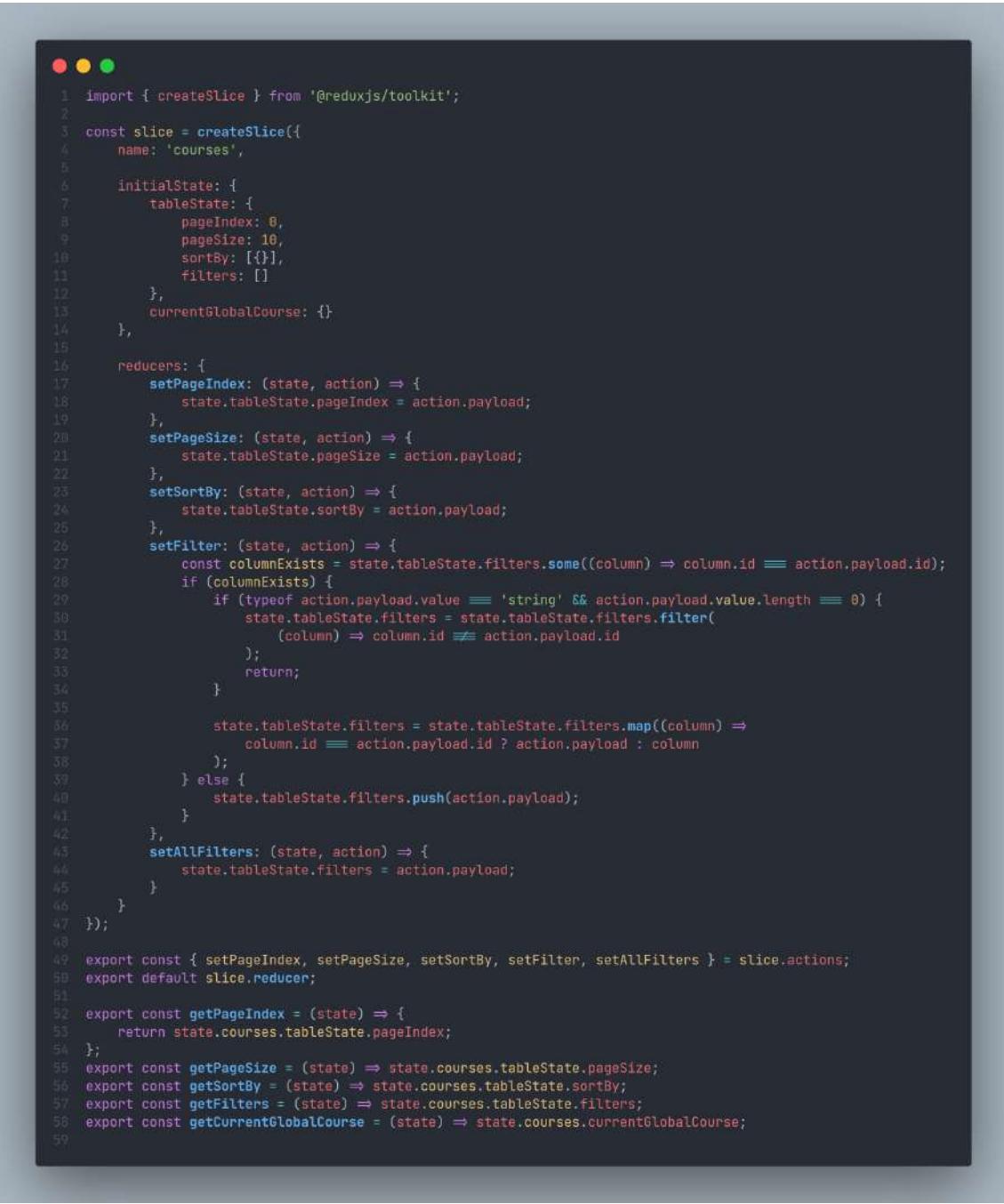
A screenshot of a code editor showing a file named 'courseSlice.js'. The code defines a Redux slice for 'courses' with reducers for setPageIndex, pageSize, sortBy, filter, and filters.

Figure 5.62: Course Slice

The course slice consists of `setPageIndex`, `setPageSize`, `setSortBy`, `setFilter`, and `setAllFilters` reducers. The `setPageIndex` reducer is responsible for setting the page index of the courses' page. The `setPageSize` reducer is responsible for setting the page size of the courses' page. The `setSortBy` reducer is responsible for setting the sort by of the courses' page. The `setFilter` reducer is responsible for setting the filter of the courses' page. The `setAllFilters` reducer is responsible for setting all of the

filters of the courses' page. The courses slice also consists of getPageIndex, getPageSize, getSortBy, getFilters, and getCurrentGlobalCourse selectors. Selectors are functions that are used to select a specific part of the state.

The courses slice's reducers are used to keep the state of the resource and then pass the values to react query to fetch the data from the backend based on the state of the resource.

The other slices are defined in a similar manner to the courses slice. In general the STDC uses these slices to manage the state of the global filters of the resource tables and pages. However, since the STDC continuously communicates with the backend, the STDC client uses Redux for its state management only when needed.

Views

The Student Talent Development Center (STDC) client (frontend) is a single page application. The STDC client is divided into components, each component is responsible for rendering a specific part of the application. The component directory is located in the src directory. The components directory is the directory that contains the components that the STDC client uses for its views. The Views directory is the directory that contains the views that the STDC client renders as a page. Mainly there are two types of views in the STDC client, the static views, and the dynamic views. The static views are the views that do not require the user to be authenticated to access them. The dynamic views are the views that require the user to be authenticated to access them. The static views are as follows:

1. LandingPage: The LandingPage component is responsible for rendering the login page.
2. NotFoundPage: The NotFoundPage component is responsible for rendering the not found page when the server returns 404.
3. UnauthorizedPage: The UnauthorizedPage component is responsible for rendering the unauthorized page when the server returns 401.
4. ForbiddenPage: The ForbiddenPage component is responsible for rendering the forbidden page when the server returns 403.

The dynamic views are as follows:

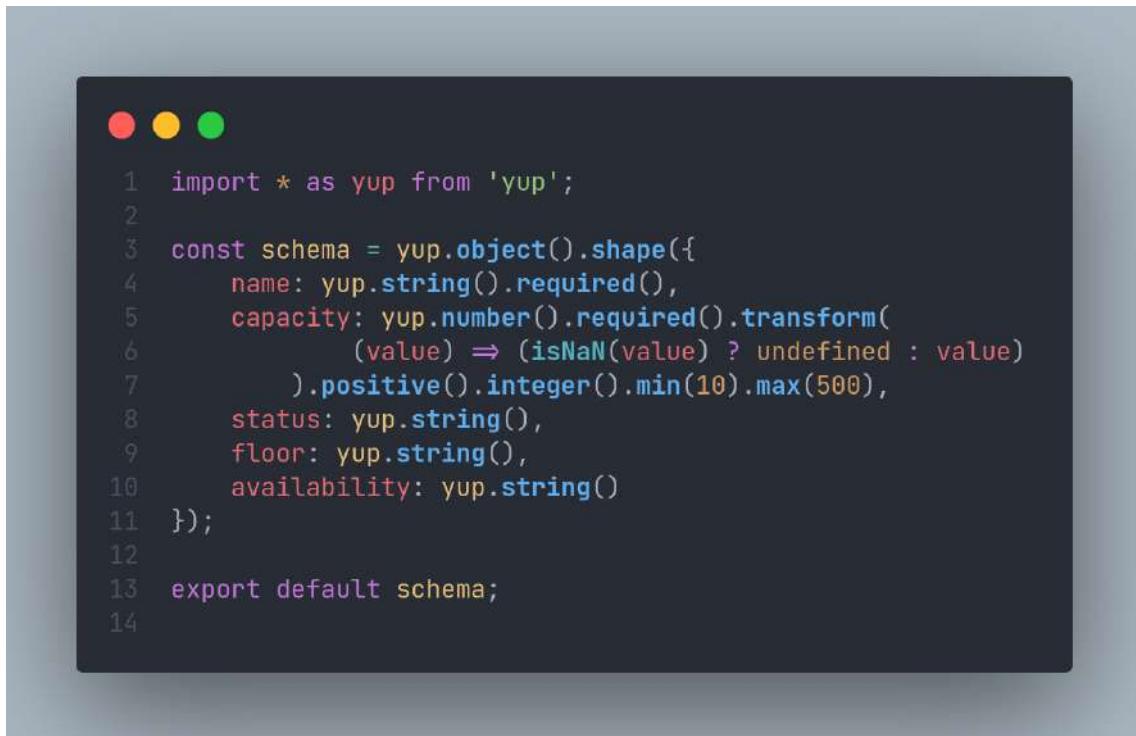
1. App: The App component is the entry point of the application. The App component is responsible for setting up the application and rendering the application.
2. NavBar: The NavBar component is responsible for rendering the top AppBar.
3. Sidebar: The Sidebar component is responsible for rendering the sidebar which contains the navigation links.
4. CoursesPage: The CoursesPage component is responsible for rendering the courses page.
5. UsersPage: The UsersPage component is responsible for rendering the users

page.

6. SessionsPage: The SessionsPage component is responsible for rendering the sessions page.
7. FeedbacksPage: The FeedbacksPage component is responsible for rendering the feedbacks page.
8. QuestionsPage: The QuestionsPage component is responsible for rendering the questions page.
9. ReleaseNotesPage: The ReleaseNotesPage component is responsible for rendering the release notes page.

Each dynamic view has their own child views. Let's take the venues view as an example. The venues view consists of the followings:

- schema: The schema is the schema to validate the venues' data on creation and update. Yup is used to validate the venues' data.



```
1 import * as yup from 'yup';
2
3 const schema = yup.object().shape({
4     name: yup.string().required(),
5     capacity: yup.number().required().transform(
6         (value) => isNaN(value) ? undefined : value)
7             .positive().integer().min(10).max(500),
8     status: yup.string(),
9     floor: yup.string(),
10    availability: yup.string()
11 });
12
13 export default schema;
14
```

Figure 5.63: Venue Schema

The venue schema consists of the name, capacity, status, floor, and availability fields. The name field is a required string field. The capacity field is a required number field. The status field is a string field. The floor field is a string field. The availability field is a string field. Each of the fields has its own validation rules. Capacity field is a number field that is positive, integer, and between 10 and 500. The schema is used to validate the venues' data on creation and update, so when the capacity field is not a number, or is not positive, or is not integer, or is not between 10 and 500, then the schema will return an error. Thus, it will not call the API, unless all the client side validations are passed.

- all: The all directory is the main Venues Page view. It consists of the following:
 - components/columnns: It is the responsibility of the columns.js file to generate the columns configuration that is utilized in the table view of the venues page. GenerateColumns is a function component that is exported from this file. It accepts setGlobalFilter, setGlobalPageIndex, and currentUser as its parameters.

Inside of the component, a number of options arrays are defined, such as statusOptions, floorOptions, and availabilityOptions. These arrays are used to store the various filtering options that can be applied to particular column types.

The component gives back an array of column objects that, collectively, define the table's columns. Each column object has properties such as Header, which is the title of the column header, and accessor, which is the data accessor for the cell value. Additionally, each column object can have optional properties such as Filter, which is the filter component for the column, Cell, which is the custom rendering component for the cell, and more.

In order to enable filtering based on particular criteria, certain columns make use of custom filter components, such as SearchColumnFilter and AutocompleteColumnFilter. When formatting date values in the createdAt

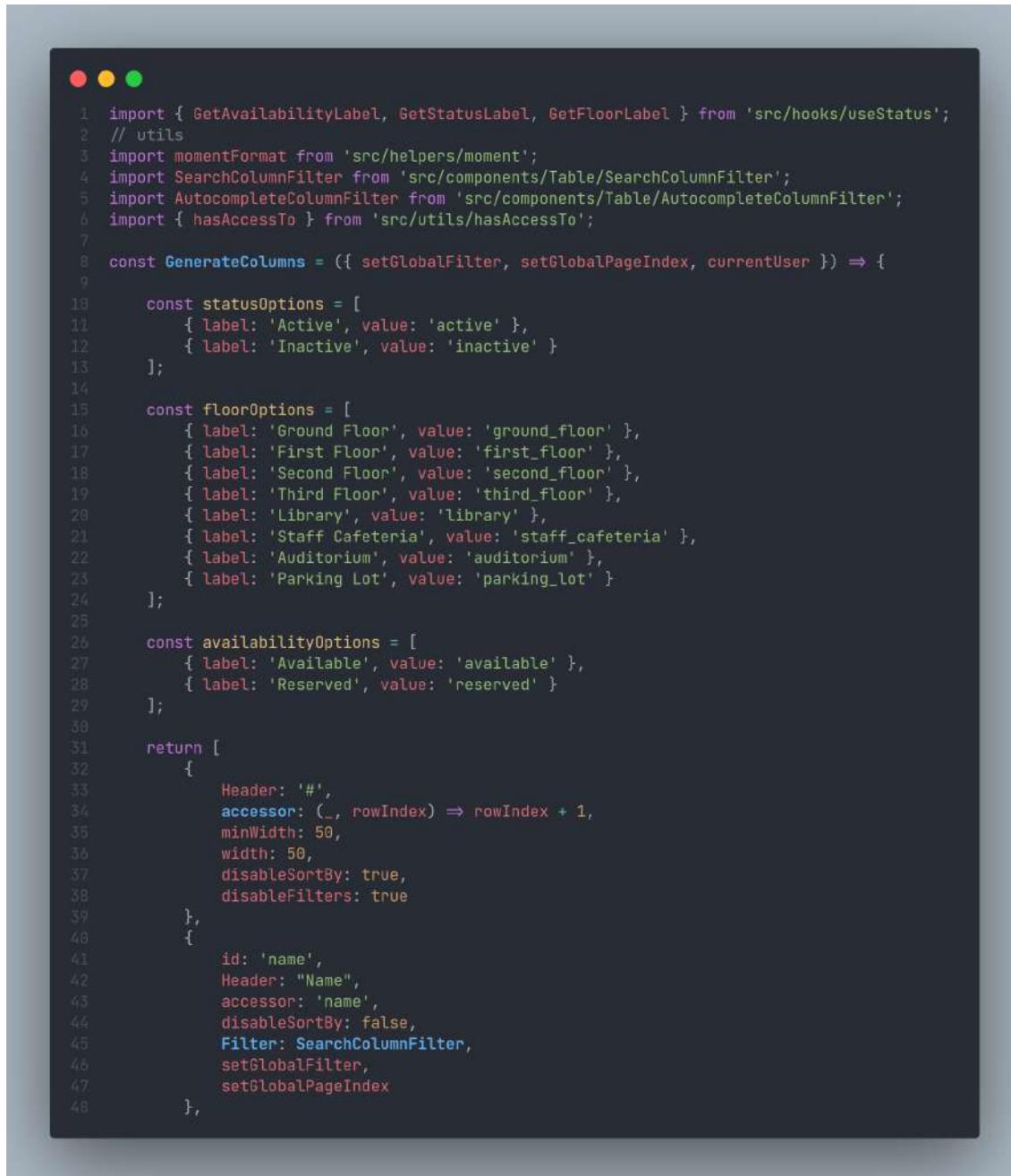
and updatedAt columns, the momentFormat helper function is called upon to do the formatting.

Finally, an additional "Actions" column is appended to the returned array of columns in a conditional form if the currently logged-in user possesses the required permissions (the "superadmin" role).

This GenerateColumns component plays an important part in defining the columns and their behaviors in the venues table. This gives users the ability to view and interact with the data in an efficient manner.

```
1 import { GetAvailabilityLabel, GetStatusLabel, GetFloorLabel } from 'src/hooks/useStatus';
2 // utils
3 import momentFormat from 'src/helpers/moment';
4 import SearchColumnFilter from 'src/components/Table/SearchColumnFilter';
5 import AutocompleteColumnFilter from 'src/components/Table/AutocompleteColumnFilter';
6 import { hasAccessTo } from 'src/utils/hasAccessTo';
7
8 const GenerateColumns = ({ setGlobalFilter, setGlobalPageIndex, currentUser }) => {
9
10    const statusOptions = [
11        { label: 'Active', value: 'active' },
12        { label: 'Inactive', value: 'inactive' }
13    ];
14
15    const floorOptions = [
16        { label: 'Ground Floor', value: 'ground_floor' },
17        { label: 'First Floor', value: 'first_floor' },
18        { label: 'Second Floor', value: 'second_floor' },
19        { label: 'Third Floor', value: 'third_floor' },
20        { label: 'Library', value: 'library' },
21        { label: 'Staff Cafeteria', value: 'staff_cafeteria' },
22        { label: 'Auditorium', value: 'auditorium' },
23        { label: 'Parking Lot', value: 'parking_lot' }
24    ];
25
26    const availabilityOptions = [
27        { label: 'Available', value: 'available' },
28        { label: 'Reserved', value: 'reserved' }
29    ];
30
31    return [
32        {
33            Header: '#',
34            accessor: (_, rowIndex) => rowIndex + 1,
35            minWidth: 50,
36            width: 50,
37            disableSortBy: true,
38            disableFilters: true
39        },
40        {
41            id: 'name',
42            Header: "Name",
43            accessor: 'name',
44            disableSortBy: false,
45            Filter: SearchColumnFilter,
46            setGlobalFilter,
47            setGlobalPageIndex
48        },
49    ];
50}
```

Figure 5.64: Venue Columns - Part 1



```
1 import { GetAvailabilityLabel, GetStatusLabel, GetFloorLabel } from 'src/hooks/useStatus';
2 // utils
3 import momentFormat from 'src/helpers/moment';
4 import SearchColumnFilter from 'src/components/Table/SearchColumnFilter';
5 import AutocompleteColumnFilter from 'src/components/Table/AutocompleteColumnFilter';
6 import { hasAccessTo } from 'src/utils/hasAccessTo';
7
8 const GenerateColumns = ({ setGlobalFilter, setGlobalPageIndex, currentUser }) => {
9
10    const statusOptions = [
11        { label: 'Active', value: 'active' },
12        { label: 'Inactive', value: 'inactive' }
13    ];
14
15    const floorOptions = [
16        { label: 'Ground Floor', value: 'ground_floor' },
17        { label: 'First Floor', value: 'first_floor' },
18        { label: 'Second Floor', value: 'second_floor' },
19        { label: 'Third Floor', value: 'third_floor' },
20        { label: 'Library', value: 'library' },
21        { label: 'Staff Cafeteria', value: 'staff_cafeteria' },
22        { label: 'Auditorium', value: 'auditorium' },
23        { label: 'Parking Lot', value: 'parking_lot' }
24    ];
25
26    const availabilityOptions = [
27        { label: 'Available', value: 'available' },
28        { label: 'Reserved', value: 'reserved' }
29    ];
30
31    return [
32        {
33            Header: '#',
34            accessor: (_, rowIndex) => rowIndex + 1,
35            minWidth: 50,
36            width: 50,
37            disableSortBy: true,
38            disableFilters: true
39        },
40        {
41            id: 'name',
42            Header: "Name",
43            accessor: 'name',
44            disableSortBy: false,
45            Filter: SearchColumnFilter,
46            setGlobalFilter,
47            setGlobalPageIndex
48        },
49    ];
50}
```

Figure 5.65: Venue Columns - Part 2

- **PageHeader.js:** The PageHeader.js file exports a component with the name PageHeader that has functional significance. It renders a Typography component with the title "Venues" as well as a conditional Button component for the purpose of creating new venues. The hasAccessTo function is used to determine whether or not the currently logged-in user possesses the "superadmin" role's required permissions to view the "create" button. By utilizing the useNavigate hook that is provided by react-

router-dom, the Button component, when it is activated, will navigate to the /venues/create route.



The screenshot shows a code editor window with a dark theme. The code is written in JavaScript and uses MUI components like Typography, Button, and Grid. It imports various hooks from 'react-router-dom' and 'src/utils'. The component, named PageHeader, contains logic to check if the user has access to create venues based on their role. If they do, it renders a 'New' button with a plus icon that, when clicked, navigates to the '/venues/create' route. The code is well-structured with clear comments and imports.

```
1 import { Typography, Button, Grid } from '@mui/material';
2 import { useNavigate } from 'react-router-dom';
3 import { hasAccessTo } from 'src/utils/hasAccessTo';
4 import { useCurrentUser } from 'src/slices/user';
5 import AddIcon from '@mui/icons-material/Add';
6
7 function PageHeader() {
8     const navigate = useNavigate();
9
10    const currentUser = useCurrentUser();
11
12    return (
13        <Grid container justifyContent="space-between" alignItems="center">
14            <Grid item>
15                <Typography variant="h4" component="h3" gutterBottom>
16                    Venues
17                </Typography>
18            </Grid>
19            <Grid item>
20                {hasAccessTo({
21                    permissions: ['superadmin'],
22                    roles: currentUser?.data?.attributes?.roles?.map((role) => role)
23                }) && (
24                    <Button
25                        startIcon={<AddIcon />}
26                        sx={{ mt: { xs: 2, md: 0 }, color: '#fff' }}
27                        color="textDark"
28                        variant="contained"
29                        onClick={() => navigate('/venues/create')}
30                    >
31                        New
32                    </Button>
33                )}
34            </Grid>
35        </Grid>
36    );
37 }
38
39 export default PageHeader;
```

Figure 5.66: Venue Page Header

- o **Tables.js:** The table view of the venues is represented by the Table.js file. This file is a React component. It imports a variety of dependencies and components that are necessary for rendering the table, such as the ReactTable component that is located in the src/components/Table/ReactTable module and the generateColumns function that is located in the ./components/columns file.

Using the useState hook, the Table component's internal storage contains a number of state variables that have been defined. These variables are used for managing the deletion and activation of items, as well as tracking the total pages and total items in the table. Additionally, they are used to manage the variables that manage the total pages.

The output of calling the generateColumns function results in the table having a wide variety of different column configurations.

In order to retrieve the data for the table, the useQuery hook was utilized. It requires an array of dependencies, which may include the currentPageIndex, currentPageSize, and filters, in addition to an asynchronous function that is responsible for retrieving the data. The data that was retrieved is saved in the variable known as data, and additional variables such as isFetching are used to monitor the loading state of the data.

When rendered, the ReactTable component receives a number of props that, collectively, supply the table with its configuration and its data. For the purposes of retrieving and updating the table state, it makes use of the generated columns array, the data that has been fetched, and functions such as getFilters, getPageIndex, getPageSize, and getSortBy. The actions prop renders buttons in a conditional manner, taking into account the permissions of the user. This makes it possible for items to be deactivated or activated.

Last but not least, the Table component renders a DeleteModal and an ActivateModal component. These components are employed for confirming and carrying out actions associated to the deactivation of data and activation of data, respectively.

The Table.js component, in whole, brings together a number of different functionalities in order to render a table view of the venues. These functionalities include data fetching, pagination, sorting, filtering, and action buttons for managing the venues' current status.

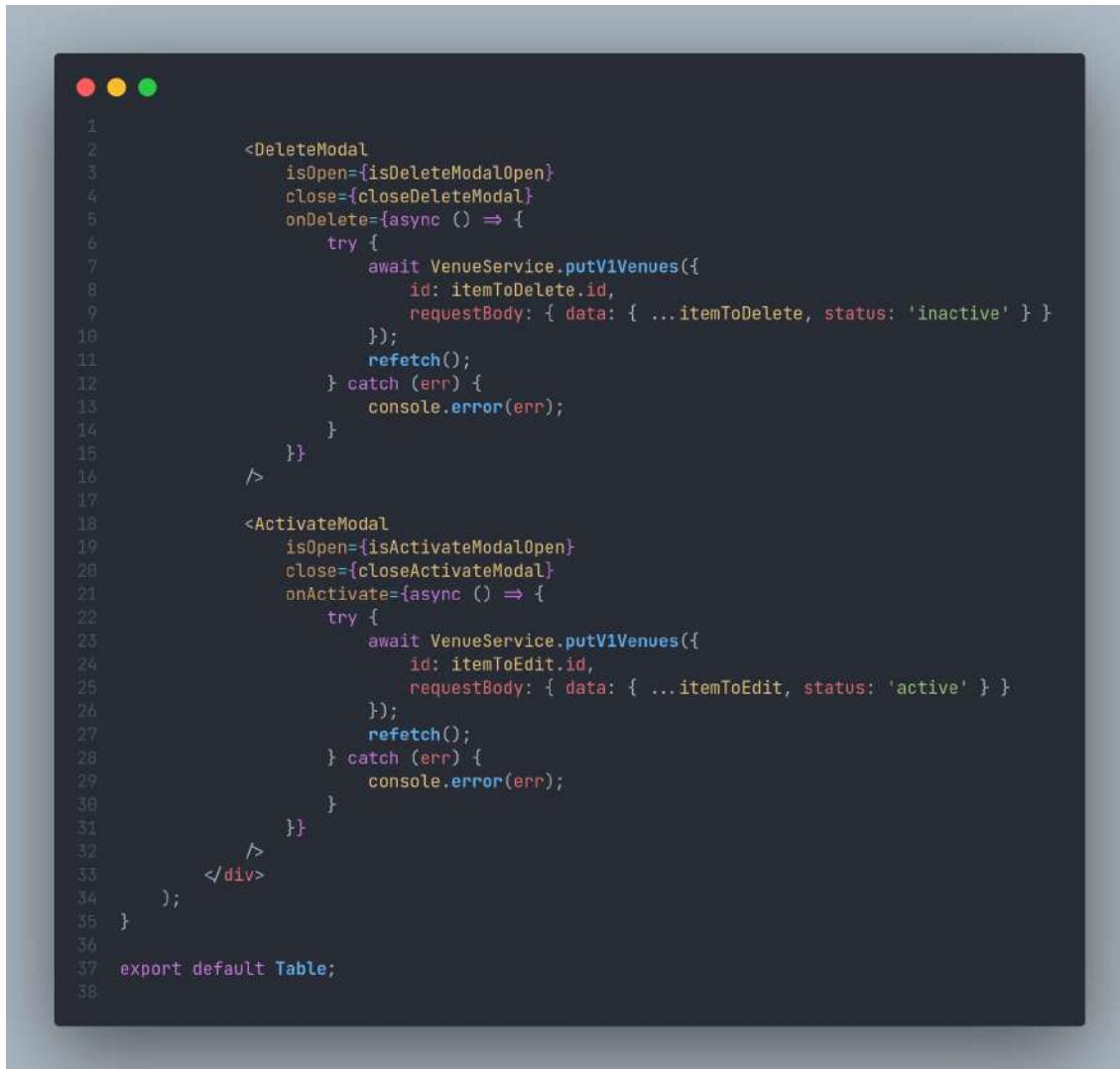
```
1  function Table() {
2      const navigate = useNavigate();
3      const currentUser = useSelector((state) => state.user.currentUser);
4
5      const [itemToDelete, setItemToDelete] = useState(null);
6      const [isDeleteModalOpen, openDeleteModal, closeDeleteModal] = useModal();
7
8      const [itemToEdit, setItemToEdit] = useState(null);
9      const [isActivateModalOpen, openActivateModal, closeActivateModal] = useModal();
10
11     const [totalPages, setTotalPages] = useState(0);
12     const [totalItems, setTotalItems] = useState(0);
13
14     const columns = generateColumns({ setGlobalFilter, setGlobalPageIndex, currentUser });
15     const filters = useSelector(getFilters);
16     const currentPageIndex = useSelector(getPageIndex);
17     const currentPageSize = useSelector(getPageSize);
18     const { errorsToast } = useToast();
19
20     const { data, isFetching, refetch } = useQuery(
21         ['venues', currentPageIndex, currentPageSize, filters],
22         async () => {
23             let normalizedFilters = normalizeFilters(filters);
24             try {
25                 const res = await VenueService.getV1Venues({
26                     status: hasAccessTo({
27                         permissions: ['superadmin'],
28                         roles: currentUser?.data?.attributes?.roles?.map((role) => role)
29                     })
30                     ? undefined
31                     : 'active',
32                     page: currentPageIndex + 1,
33                     limitValue: currentPageSize,
34                     ...normalizedFilters
35                 });
36                 setTotalPages(res?.meta?.page?.totalPages);
37                 setTotalItems(res?.meta?.page?.count);
38
39                 return res.data;
40             } catch (err) {
41                 errorsToast(err);
42             }
43         },
44         { enabled: true, keepPreviousData: true, refetchOnWindowFocus: false, initialData: [] }
45     );
46 }
```

Figure 5.67: Venue Table - Part 1

```
1
2 return (
3     <div>
4         <ReactTable
5             columns={columns}
6             data={data}
7             getFilters={getFilters}
8             getPageIndex={getPageIndex}
9             getPageSize={getPageSize}
10            getSortBy={getSortBy}
11            loading={isFetching}
12            setAllGlobalFilters={setAllGlobalFilters}
13            setGlobalPageIndex={setGlobalPageIndex}
14            setGlobalPageSize={setGlobalPageSize}
15            setGlobalSortBy={setGlobalSortBy}
16            totalItems={totalItems}
17            totalPages={totalPages}
18            handleRowClick={(row) =>
19                checkPermission(currentUser, 'superadmin') && navigate(`/venues/edit/${row.original.id}`)
20            }
21            enableCard={true}
22            showFiltersOnHeader={false}
23            enhancedFilter
24            actions={(
25                hasAccessTo({
26                    permissions: ['superadmin'],
27                    roles: currentUser?.data?.attributes?.roles?.map((role) => role)
28                })
29                ? (row) =>
30                    row.status === 'active' ? (
31                        <LoadingButton
32                            variant="outlined"
33                            size="small"
34                            color="error"
35                            onClick={() => {
36                                setItemToDelete(row);
37                                openDeleteModal();
38                            }}
39                        >
40                            {'Deactivate'}
41                        </LoadingButton>
42                    ) : (
43                        <LoadingButton
44                            variant="outlined"
45                            size="small"
46                            color="success"
47                            onClick={() => {
48                                setItemToDelete(row);
49                                openActivateModal();
50                            }}
51                        >
52                            {'Activate'}
53                        </LoadingButton>
54                    )
55                : undefined
56            })
57        />

```

Figure 5.68: Venue Table - Part 2



```
1      <DeleteModal
2        isOpen={isDeleteModalOpen}
3        close={closeDeleteModal}
4        onDelete={async () =>
5          try {
6            await VenueService.putV1Venues({
7              id: itemToDelete.id,
8              requestBody: { data: { ...itemToDelete, status: 'inactive' } }
9            );
10           refetch();
11         } catch (err) {
12           console.error(err);
13         }
14       }
15     />
16
17     <ActivateModal
18       isOpen={isActivateModalOpen}
19       close={closeActivateModal}
20       onActivate={async () =>
21         try {
22           await VenueService.putV1Venues({
23             id: itemToEdit.id,
24             requestBody: { data: { ...itemToEdit, status: 'active' } }
25           );
26           refetch();
27         } catch (err) {
28           console.error(err);
29         }
30       }
31     />
32   </div>
33 }
34 )
35
36 export default Table;
37
```

Figure 5.69: Venue Table - Part 3

- **Index.js:** The `Index.js` file is the starting point for the page that contains the venues. It imports the `PageHeader` component and the `Table` component, and then renders them inside of a container. The `PageTitleWrapper` component surrounds the `PageHeader` component, which enables it to provide a formatted page title section. In order to guarantee that the layout is executed correctly, the `Table` component is nested within a `Grid` item.



```
1 import PageTitleWrapper from 'src/components/PageTitleWrapper';
2 import { Container, Grid } from '@mui/material';
3 import PageHeader from './PageHeader';
4 import Table from './Table';
5
6 function Index() {
7     return (
8         <>
9             <PageTitleWrapper maxWidth="xl">
10                 <PageHeader />
11             </PageTitleWrapper>
12             <Container maxWidth="xl">
13                 <Grid container direction="row" justifyContent="center"
14                     alignItems="stretch" spacing={3}>
15                     <Grid item xs={12}>
16                         <Table />
17                     </Grid>
18                 </Grid>
19             </Container>
20         </>
21     );
22 }
23
24 export default Index;
25
```

Figure 5.70: Venue Index

- **create:** A new venue can be created by filling out the form that is represented by the Create component, which is a React component. It gets two props, called `defaultNode` and `onSuccess`, when it succeeds. The `onSuccess` prop is a callback function that is to be executed after a successful creation of a new venue, and the `defaultNode` prop is used to determine whether the form is being used in the context of creating a new venue or editing an existing one.

Within the component itself, a number of different dependencies and components are imported. These include `PageTitle`, `useForm`, and `yupResolver`, in addition to several components derived from Material-UI.

By utilizing the `useState` hook, the component creates state variables. One of these variables, `btnLoading`, is used to monitor the loading state of the submit button.

It is possible to retrieve functions for displaying success and error toasts by making use of the `useToast` hook.

In addition to this, the component makes use of the `useForm` hook in order to initialize a form with a validation schema that is resolved by `yupResolver`.

The `useMemo` hook is utilized in the definition of the `fields` variable, which is responsible for storing an array of objects that correspond to the form fields. Each object has its own set of properties that can be used to configure the form field. These properties include `fieldType`, `name`, `label`, `control`, and `props`.

The asynchronous `handleFormSubmit` function is the function that is called whenever the form is submitted for processing. Through the use of the `VenueService.postV1Venues` method, it submits a request to the server in order to create a new venue. In the event that the request is processed without error, a success toast will be shown to the user, and either the `onSuccess` callback function will be carried out or the user will be taken to the `/venues` page.

Within the context of the `Card` component, the component renders a form. The `FormFields` component comes from the `mui-rhf-library` and is a part of the form. It is responsible for rendering the form fields according to the `fields` configuration.

In addition, there are buttons on the form that allow users to navigate back through the form and submit their responses. Clicking the submit button will cause the `handleFormSubmit` function to be triggered.

In general, the `Create` component is the form that users fill out in order to create a new venue. Additionally, it offers functionality that manages the submission of forms and navigation.

```
1 import PageTitle from 'src/components/PageTitle/PageHeader';
2 import { useMemo, useState } from 'react';
3 import { useForm } from 'react-hook-form';
4 import { yupResolver } from '@hookform/resolvers/yup';
5 import PageTitleWrapper from 'src/components/PageTitleWrapper';
6 import { Container, Grid, Card, CardContent } from '@mui/material';
7 import schema from '../schema';
8 import { useNavigate } from 'react-router-dom';
9 import { FormFields } from 'mui-rhf-library';
10 import { VenueService } from 'src/services/api/stdc';
11 import { useToast } from 'src/hooks/useToast';
12 import LoadingButton from 'src/components/Button>LoadingButton';
13 import { Stack } from '@mui/system';
14 import { BasicBreadcrumbs } from 'src/components/BasicBreadcrumbs/BasicBreadcrumbs';
15
16 function Create({ defaultNode, onSuccess }) {
17   const navigate = useNavigate();
18   const [btnLoading, setBtnLoading] = useState(false);
19
20   const { errorsToast, successToast } = useToast();
21
22   const { handleSubmit, control } = useForm({
23     resolver: yupResolver(schema)
24   });

```

Figure 5.71: Create Venue - Part 1

```
 1 const fields = useMemo(() => {
 2   return [
 3     {
 4       fieldType: 'textField',
 5       name: 'name',
 6       label: "Name",
 7       control: control,
 8       props: { fullWidth: true },
 9       gridProps: { xs: 12, md: 6 }
10     },
11     {
12       fieldType: 'textField',
13       name: 'capacity',
14       label: "Capacity",
15       control: control,
16       props: { fullWidth: true, type: 'number' },
17       gridProps: { xs: 12, md: 6 }
18     },
19     {
20       name: 'floor',
21       label: 'Floor',
22       fieldType: 'select',
23       props: {
24         options: [
25           { value: 'ground_floor', label: 'Ground Floor' },
26           { value: 'first_floor', label: 'First Floor' },
27           { value: 'second_floor', label: 'Second Floor' },
28           { value: 'third_floor', label: 'Third Floor' },
29           { value: 'library', label: 'Library' },
30           { value: 'staff_cafeteria', label: 'Staff Cafeteria' },
31           { value: 'auditorium', label: 'Auditorium' },
32           { value: 'parking_lot', label: 'Parking Lot' }
33         ],
34         fullWidth: true
35       },
36       gridProps: { xs: 12, md: 6 }
37     },
38     {
39       name: 'availability',
40       label: "Availability",
41       fieldType: 'select',
42       props: {
43         options: [
44           { value: 'available', label: 'Available' },
45           { value: 'reserved', label: 'Reserved' },
46         ],
47         fullWidth: true
48       },
49       gridProps: { xs: 12, md: 6 },
50       defaultValue: 'available'
51     },
52   ];
53 }, []);
54
55 const handleFormSubmit = async (data) => {
56   setBtnLoading(true);
57   try {
58     await VenueService.postV1Venues({
59       requestBody: { data: { ...data } }
60     });
61     successToast("Venue created successfully!");
62     onSuccess ? onSuccess() : navigate('/venues');
63   } catch (err) {
64     errorsToast(err);
65   } finally {
66     setBtnLoading(false);
67   }
68};
```

Figure 5.72: Create Venue - Part 2

```
1  return (
2    <PageTitleWrapper maxWidth="xl">
3      <BasicBreadcrumbs
4        links={[
5          {
6            label: 'Venues',
7            href: '/venues'
8          }
9        ]}
10       currentLinkLabel={'New Venue'}
11     />
12
13     <PageTitle heading="New Venue" />
14   </PageTitleWrapper>
15   <Container maxWidth="xl">
16     <Grid container direction="row" justifyContent="center"
17       alignItems="stretch" spacing={3}>
18
19       <Grid item xs={12}>
20         <Card variant="outlined" sx={{ borderRadius: 2 }}>
21           <CardContent>
22             <form onSubmit={handleSubmit(handleFormSubmit)}>
23               <Grid container spacing={2}>
24                 <FormFields fields={fields} control={control} />
25
26                 <Grid item xs={12}>
27                   <Stack direction="row" spacing={2}>
28                     {!defaultNode && (
29                       <LoadingButton
30                         loading={btnLoading}
31                         onClick={() => {
32                           navigate(-1);
33                         }}
34                         variant="outlined"
35                         color="textDark"
36                       >
37                         Back
38                       </LoadingButton>
39                     )}
40                     <LoadingButton
41                         color="textDark"
42                         loading={btnLoading}
43                         type="submit"
44                         variant="contained"
45                         sx={{ color: '#fff' }}
46                       >
47                         Submit
48                       </LoadingButton>
49                     </Stack>
50                   </Grid>
51                 </form>
52               </CardContent>
53             </Card>
54           </Grid>
55         </Grid>
56       </Container>
57     </>
58   );
59 }
60
61 export { Create };
62
63
```

Figure 5.73: Create Venue - Part 3

- edit: The Create component and the Edit component have some similarities in common, but the Edit component also has some distinctive differences. To begin, the `getProfile` function is utilized within the Edit component in order to retrieve the already existing venue data from the server based on the `id` parameter that is provided. After that, the data is saved into the variable that represents the venue's state. Second, when the form fields are being defined in the Edit component, the default values are set based on the retrieved venue data. This makes it possible for the form to be pre-filled with the information that already exists. In addition, the `handleFormSubmit` function found in the Edit component is responsible for performing a PUT request to update the venue by utilizing the `VenueService.putV1Venues` method. This request includes the updated form data as well as the `id` parameter. The cancel button triggers the navigation function, which takes the user to the previous page. Last but not least, the Edit component will render an empty div if the venue data has not yet been made available. This ensures that the form will not be rendered without the essential data.



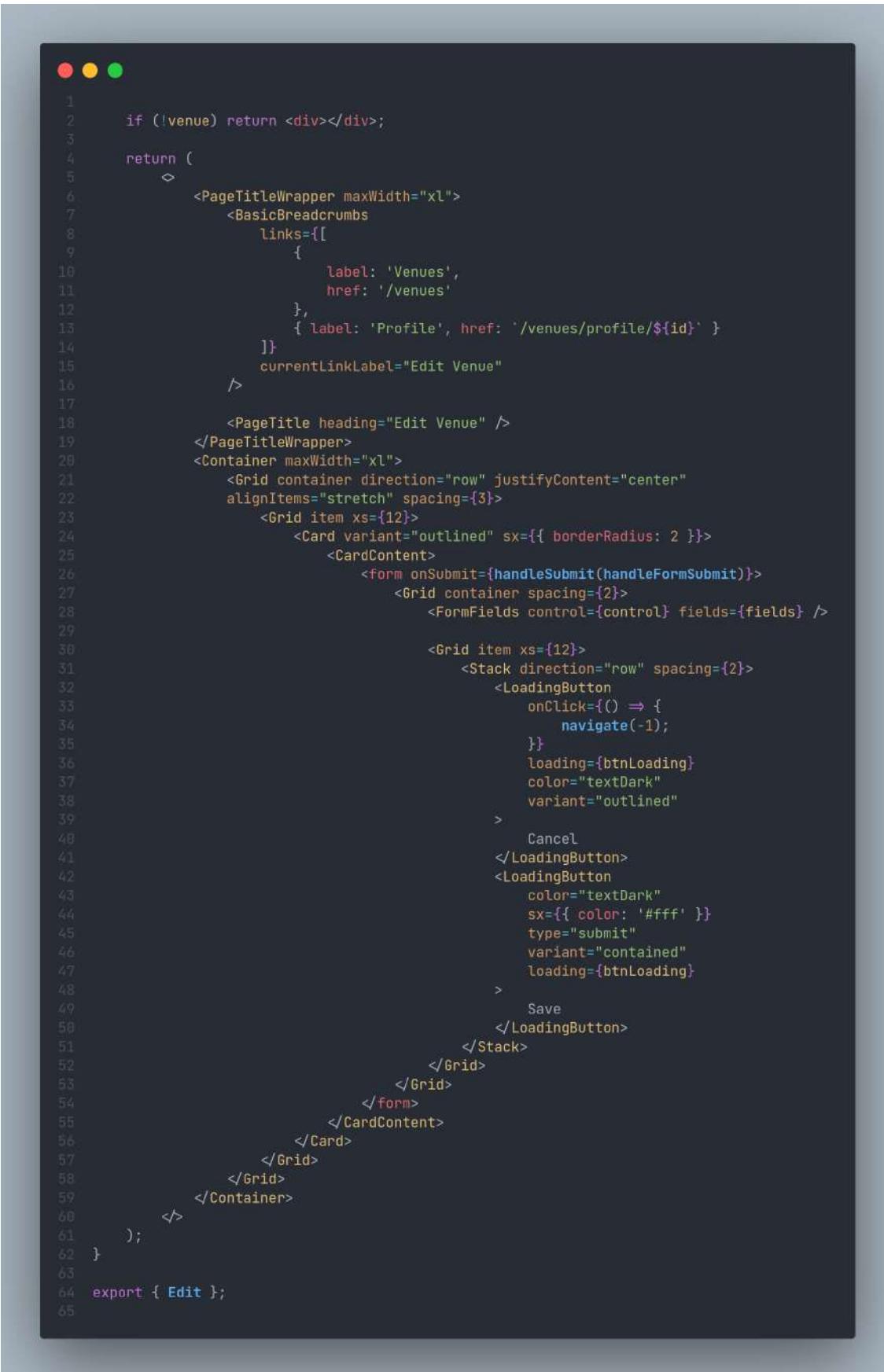
```
1 import PageTitle from 'src/components/PageTitle/PageHeader';
2 import React, { useState, useCallback, useEffect, useMemo } from 'react';
3 import { useForm } from 'react-hook-form';
4 import { yupResolver } from '@hookform/resolvers/yup';
5 import PageTitleWrapper from 'src/components/PageTitleWrapper';
6 import { Container, Grid, Card, CardContent } from '@mui/material';
7 import schema from '../schema';
8 import { useNavigate, useParams } from 'react-router-dom';
9 import { VenueService } from 'src/services/api/stdc';
10 import { useToast } from 'src/hooks/useToast';
11 import { FormFields } from 'mui-rhf-library';
12 import { Stack } from '@mui/system';
13 import LoadingButton from 'src/components/Button>LoadingButton';
14 import { BasicBreadcrumbs } from 'src/components/BasicBreadcrumbs/BasicBreadcrumbs';
15
16 function Edit() {
17   const navigate = useNavigate();
18   const { errorsToast, successToast } = useToast();
19   const [btnLoading, setBtnLoading] = useState(false);
20   const { id } = useParams();
21
22   const [venue, setVenue] = useState(null);
23
24   const getProfile = useCallback(async (id) => {
25     const { data } = await VenueService.getV1Venues1({ id });
26     setVenue(data);
27   }, []);
28
29   const { handleSubmit, control } = useForm({
30     resolver: yupResolver(schema)
31   });
32
33   useEffect(() => {
34     getProfile(id);
35   }, []);
}
```

Figure 5.74: Update Venue - Part 1

```
● ● ●
1 const fields = useMemo(() => {
2   return [
3     {
4       fieldType: 'textField',
5       name: 'name',
6       label: 'Name',
7       control: control,
8       props: {
9         fullWidth: true, defaultValue: venue ? venue?.name : null
10      }, gridProps: { xs: 12, md: 6 }
11    },
12    {
13      fieldType: 'textField',
14      name: 'capacity',
15      label: 'Capacity',
16      control: control,
17      props: {
18        fullWidth: true,
19        type: 'number', defaultValue: venue ? venue?.capacity : null
20      }, gridProps: { xs: 12, md: 6 }
21    },
22    {
23      name: 'floor',
24      label: 'Floor',
25      fieldType: 'select',
26      props: {
27        options: [
28          { value: 'ground_floor', label: 'Ground Floor' },
29          { value: 'first_floor', label: 'First Floor' },
30          { value: 'second_floor', label: 'Second Floor' },
31          { value: 'third_floor', label: 'Third Floor' },
32          { value: 'library', label: 'Library' },
33          { value: 'staff_cafeteria', label: 'Staff Cafeteria' },
34          { value: 'auditorium', label: 'Auditorium' },
35          { value: 'parking_lot', label: 'Parking Lot' }
36        ],
37        fullWidth: true, defaultValue: venue?.floor
38      }, gridProps: { xs: 12, md: 6 }
39    },
40    {
41      name: 'availability',
42      label: 'Availability',
43      fieldType: 'select',
44      props: {
45        options: [
46          { value: 'available', label: 'Available' },
47          { value: 'reserved', label: 'Reserved' }
48        ],
49        fullWidth: true, defaultValue: venue?.availability
50      }, gridProps: { xs: 12, md: 6 }
51    },
52    {
53      name: 'status',
54      label: 'Status',
55      fieldType: 'select',
56      props: {
57        options: [
58          { value: 'active', label: 'Active' },
59          { value: 'inactive', label: 'Inactive' }
60        ],
61        fullWidth: true, defaultValue: venue?.status
62      }, gridProps: { xs: 12, md: 6 }
63    };
64  }, [venue]);
65
66 const handleFormSubmit = async (data) => {
67   setBtnLoading(true);
68   try {
69     await VenueService.putV1Venues({
70       id, requestBody: { data: { ...data } }
71     });
72     successToast('Venue updated successfully!');
73     navigate(-1);
74   } catch (err) {
75     errorsToast(err);
76   } finally {
77     setBtnLoading(false);
78   }
79 };

```

Figure 5.75: Update Venue - Part 2



A screenshot of a code editor displaying a React component named `Edit`. The component handles the update of a venue. It starts by checking if a venue is provided; if not, it returns a simple `<div></div>` element. If a venue is provided, it renders a `<PageTitleWrapper>` component with a `maxWidth="xl"` prop. Inside this wrapper, a `<BasicBreadcrumbs>` component is used to display a breadcrumb trail. The `links` prop for `BasicBreadcrumbs` contains two items: a link to the 'Venues' page and a link to the 'Profile' page of the current venue. The `currentLinkLabel` prop is set to 'Edit Venue'. Below the breadcrumb, a `<PageTitle heading="Edit Venue" />` component is rendered. The main content area is enclosed in a `<Container maxWidth="xl">`. Inside the container, a `<Grid direction="row" justifyContent="center" alignItems="stretch" spacing={3}>` grid is used. The first item in the grid is a `<Card variant="outlined" sx={{ borderRadius: 2 }}>` card. The card's `CardContent` contains a `<form onSubmit={handleSubmit(handleFormSubmit)}>` form. The form includes a `<Grid container spacing={2}>` grid for fields, a `<FormFields control={control} fields={fields} />` component, and a `<Grid item xs={12}>` grid for buttons. This inner grid contains a `<Stack direction="row" spacing={2}>` stack of buttons. The first button is a `<LoadingButton onClick={() => { navigate(-1); }} loading={btnLoading} color="textDark" variant="outlined">` button labeled 'Cancel'. The second button is a `<LoadingButton color="textDark" sx={{ color: '#fff' }} type="submit" variant="contained" loading={btnLoading}>` button labeled 'Save'. Both buttons have their `loading` state controlled by the `btnLoading` variable. The entire card content is enclosed in a `<Grid item xs={12}>` grid. The component concludes with an `</Grid>`, an `</Container>`, and finally an `</>` followed by a closing brace `};`. The file ends with an `export { Edit };` statement.

```
1  if (!venue) return <div></div>;
2
3  return (
4      <>
5          <PageTitleWrapper maxWidth="xl">
6              <BasicBreadcrumbs
7                  links={[
8                      {
9                          label: 'Venues',
10                         href: '/venues'
11                     },
12                     { label: 'Profile', href: `/venues/profile/${id}` }
13                 ]}
14             currentLinkLabel="Edit Venue"
15         />
16
17         <PageTitle heading="Edit Venue" />
18     </PageTitleWrapper>
19     <Container maxWidth="xl">
20         <Grid container direction="row" justifyContent="center"
21             alignItems="stretch" spacing={3}>
22             <Grid item xs={12}>
23                 <Card variant="outlined" sx={{ borderRadius: 2 }}>
24                     <CardContent>
25                         <form onSubmit={handleSubmit(handleFormSubmit)}>
26                             <Grid container spacing={2}>
27                                 <FormFields control={control} fields={fields} />
28
29                             <Grid item xs={12}>
30                                 <Stack direction="row" spacing={2}>
31                                     <LoadingButton
32                                         onClick={() => {
33                                             navigate(-1);
34                                         }}
35                                         loading={btnLoading}
36                                         color="textDark"
37                                         variant="outlined"
38                                     >
39                                         Cancel
40                                     </LoadingButton>
41                                     <LoadingButton
42                                         color="textDark"
43                                         sx={{ color: '#fff' }}
44                                         type="submit"
45                                         variant="contained"
46                                         loading={btnLoading}
47                                     >
48                                         Save
49                                     </LoadingButton>
50                                 </Stack>
51                             </Grid>
52                         </Grid>
53                     </CardContent>
54                 </Card>
55             </Grid>
56         </Container>
57     </>
58 );
59 }
60
61
62 }
63
64 export { Edit };
65
```

Figure 5.76: Update Venue - Part 3

The rest of the view pages follow the same structure, however due to the length of these views, it is not possible to discuss them in this section. Please refer to the source code for more details.

Results

In this section, we will demonstrate the completed product, which will highlight the outcome of all of the labor as well as the developments that have been incorporated. We have implemented a number of improvements throughout the course of the extensive development process in order to produce an application that is aesthetically pleasing and user-friendly. The following figures highlight the key features and design elements of the final product, demonstrating the commitment to providing the students and users of the STDC Application with the best possible experience possible.

It is worth mention that the below figure have been taken from deployed version of the STDC application available at <https://stdc.onrender.com>

Login Pages

When the user first visits the STDC Application, they are greeted with a login page. In order to login user needs to use their microsoft account. The login page is shown in the figure below.

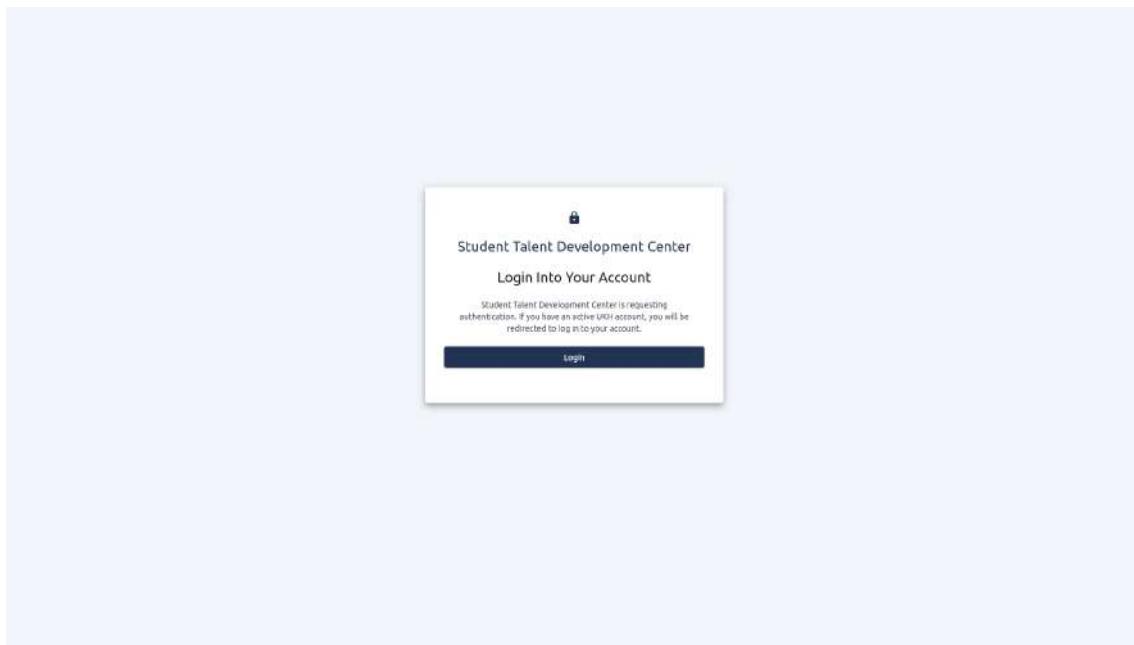


Figure 5.77: Login - Landing Page

Once the user clicks on the login button from the landing page, they are redirected to another page asking them to continue the login with a microsoft account. This page is the Universal Login page provided by the Auth0.

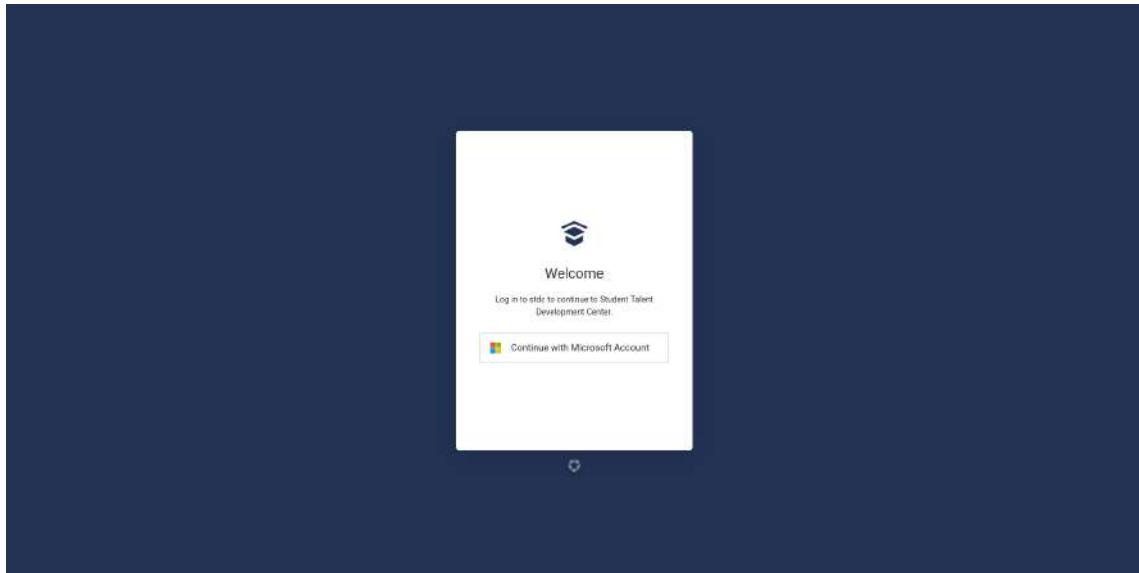


Figure 5.78: Login - Auth0 Universal Login Page

Once the user clicks on the continue with microsoft button, they are redirected to the microsoft login page. User should sign in using their already existing microsoft account.

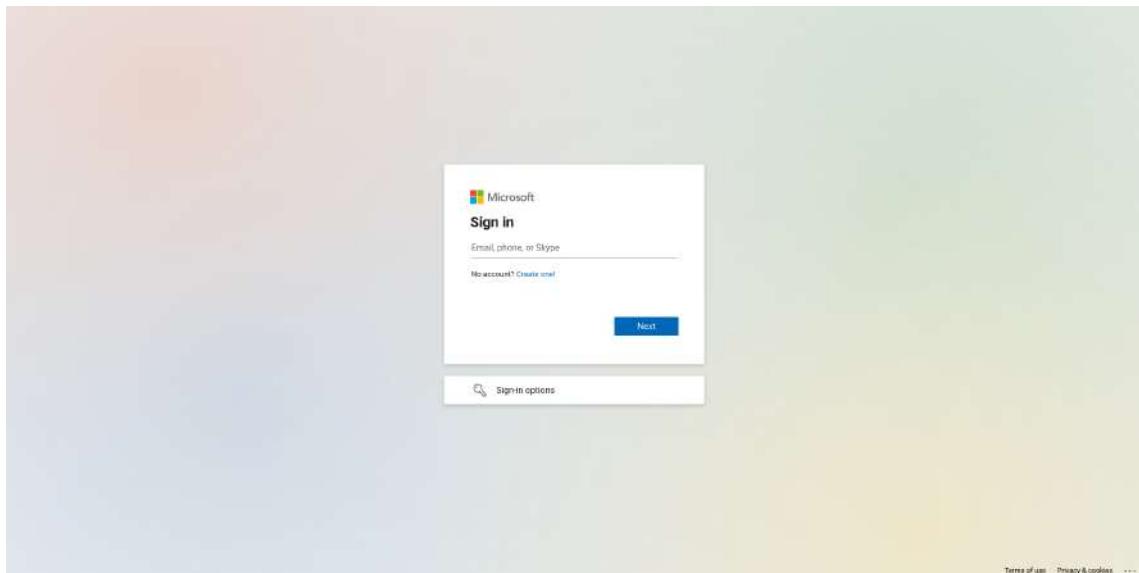


Figure 5.79: Login - Microsoft Login Page

After signing into the microsoft account, if it is the first time the user is logging in, they are asked to grant permissions to the application to access their profile information. This is a one time process and the user will not be asked to grant permissions again. Note that, after granting permissions, the user will be redirected to the home page of the application.

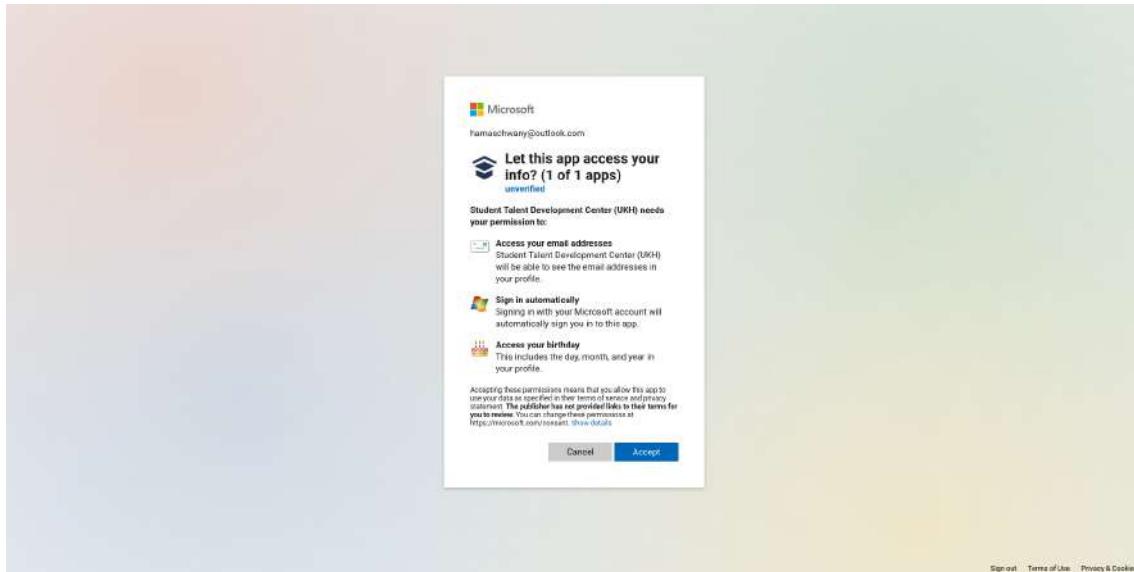


Figure 5.80: Login - Grant Permissions

Home Page

Once the user is logged in, they are introduced to the home page of the application. In the home page, the available courses of the center are shown to the user. Note that, when a superadmin views this page, the inactive courses are also shown. Another, important thing to note is that, the enrolled courses of the user are shown at the top of the page if they are available for the user.

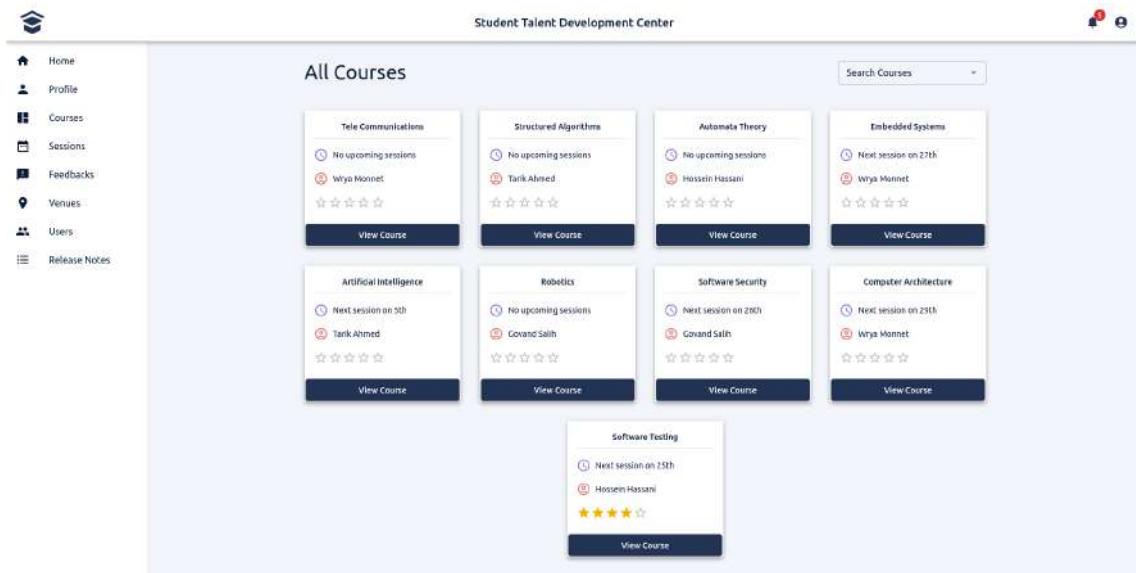


Figure 5.81: Home Page

The below figure shows the home page in the perspective of a viewer user. The sidebar elements are already filtered based on the user's role as well as the deactivated courses are not shown to the user. More on Viewer's perspective will be talked about in the coming subsections.

The screenshot shows the 'All Courses' section of the viewer perspective. It displays eight course cards arranged in two rows of four. Each card includes the course name, session status, instructor, and a star rating.

Course	Session Status	Instructor	Rating
Structured Algorithms	No upcoming sessions	Tariq Ahmed	4 stars
Automata Theory	No upcoming sessions	Hossein Hassani	3 stars
Embedded Systems	Next session on 27th	Wrya Monnet	4 stars
Artificial Intelligence	Next session on 5th	Tariq Ahmed	3 stars
Robotics	No upcoming sessions	Govind Salihi	5 stars
Software Security	Next session on 26th	Govind Salihi	4 stars
Computer Architecture	Next session on 29th	Wrya Monnet	4 stars
Software Testing	Next session on 25th	Hossein Hassani	5 stars

Footer: University of Khudan Hudaib [All Rights Reserved © 2023] | Student Talent Development Center | Mohammad Nawaz | Admin

Figure 5.82: Home Page - Viewer Perspective

Users Page

Users page is the page where the superadmin can view all the users of the application and manage the users. The below page is the users table, and one of the filters is used to filter users by their role.

The screenshot shows the 'Users' table on the users page. The table lists five users with columns for Name, Roles, Email, Status, Created At, and Actions.

#	Name	Roles	Email	Status	Created At	Actions
1	Tariq Ahmed	Tutor	tariqahmed@ukh.edu.krd	Active	25-6-2023	<button>Edit</button>
2	Wrya Monnet	Tutor	wrya.monnet@ukh.edu.krd	Active	25-6-2023	<button>Edit</button>
3	Govind Salihi	Tutor	govind.salihi@ukh.edu.krd	Active	25-6-2023	<button>Edit</button>
4	Hossein Hassani	Tutor	hossein.hassani@ukh.edu.krd	Active	25-6-2023	<button>Edit</button>
5	Mohammad Nawaz	Superadmin / Tutor	hamanshawnawaz@outlook.com	Active	8-6-2023	<button>Edit</button>

Filter Options: Name, Roles (Tutor), Status (Active), Reset

Figure 5.83: Users Page - Users Table

From users' page, the admin can onboard users by clicking the onboard button. When the page loads, the admin can select the student that they want to onboard as

a tutor.

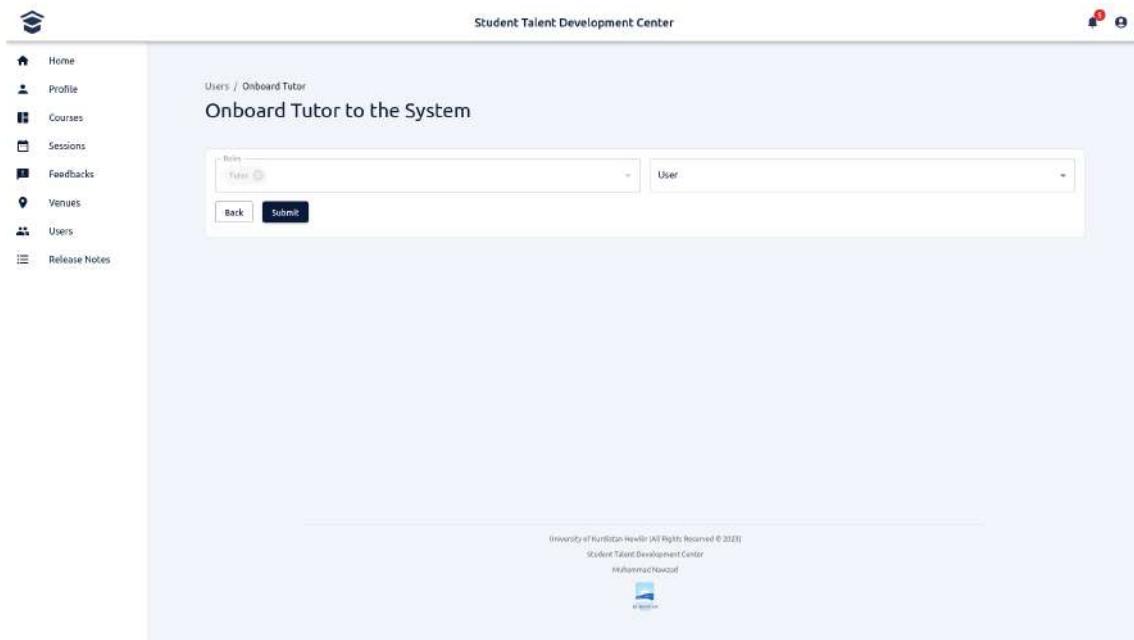


Figure 5.84: Users - Onboard Page

The admin can also view the details of a user by clicking on the rows of that specific user in the table. This will redirect to the profile page of that user. Depending on the role of the user, the profile page might look different. Below here are some of those as an example.

A screenshot of the "Profile" page for a user named "Didam Goran". The left sidebar is identical to Figure 5.84. The main content area shows a profile picture placeholder, the name "Didam Goran", the email "didam.goran@ukh.edu.krd", and status indicators "active" and "student". Below this is a "Personal Information" table:

Figure 5.85: Users - Student Profile Page

The screenshot shows the 'Profile' section of the Student Talent Development Center. On the left is a sidebar with icons for Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main area displays a profile card for 'Govand Salih' with the email 'govand.salih@ukh.edu.krd'. Below the card are two boxes: one orange showing '\$0 Payment' and one light blue showing '0 Hr Hours'. A table titled 'Personal Information' lists details: First Name (Govand), Last Name (Salih), Email (govand.salih@ukh.edu.krd), Role (tutor), and Status (active). A 'Tutor Availability' section shows a table with columns for 'Days of the Week', 'Available From', and 'Available Until'. The table lists availability from Sunday to Thursday.

Figure 5.86: Users - Tutor Profile Page - 1

This screenshot shows a different view of the tutor's profile. It includes a table with rows for 'Role' (tutor) and 'Status' (active). Below this is a 'Tutor Availability' section with a table showing weekly availability. The table has columns for 'Days of the Week', 'Available From', and 'Available Until'. The availability is listed as follows: Sunday (08:00 - 12:00), Monday (Not Available), Tuesday (Not Available), Wednesday (10:00 - 17:00), and Thursday (08:00 - 16:00). At the bottom of the page, there is a footer with copyright information: 'University of Kurdistan Hewlêr (All Rights Reserved © 2022)', 'Student Talent Development Center', 'Muhammed Nawzad', and a logo for 'Al-Mutanabbi'.

Figure 5.87: Users - Tutor Profile Page - 2

The screenshot shows a web-based application interface for the "Student Talent Development Center". The top navigation bar includes icons for Home, Profile, Sessions, and Release Notes, along with a search bar and user account information. The main content area is titled "Profile" and displays a user card for "Muhammad Nawzad" with the email "hamal27n@outlook.com". A green status indicator bar shows "active" and "viewer". Below the card is a "Personal Data" section containing a table with the following data:

Property	Information
First Name	Muhammad
Last Name	Nawzad
Email	hamal27n@outlook.com
Role	viewer
Status	active

At the bottom of the page, there is a footer with copyright information: "University of Kurdistan Hewler [All Rights Reserved © 2021]", "Student Talent Development Center", and "Muhammad Nawzad".

Figure 5.88: Users - Viewer Profile Page

Venues Page

Venues page is the page where the superadmin can view all the venues of the application and manage the venues. The below page is the venues table. Venues do not have their individual page, thus when clicking on a row, it will navigate to the edit page.

#	Name	Capacity	Status	Floor	Availability	Created At	Updated At	Actions
1	S10	25	Active	Second Floor	Available	25-6-2023	25-6-2023	<button>Deactivate</button>
2	F11	30	Active	First Floor	Available	25-6-2023	25-6-2023	<button>Deactivate</button>
3	T10	10	Active	Third Floor	Available	25-6-2023	25-6-2023	<button>Deactivate</button>
4	Auditorium	500	Active	Ground Floor	Reserved	25-6-2023	25-6-2023	<button>Deactivate</button>
5	L1	10	Inactive	Ground Floor	Reserved	25-6-2023	25-6-2023	<button>Activate</button>

Figure 5.89: Venues Page - Venues Table

From venues' page, the admin can add new venues by clicking the new button. This will redirect to create venue page where you can add new venues. The below figures show the create venue page normally, then when the required fields are not provided, basically shows the client side validation. Other creation pages are also similar to this one.

The screenshot shows the 'New Venue' creation page. On the left is a sidebar with navigation links: Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main content area has a header 'Student Talent Development Center' and a breadcrumb 'Venues / New Venue'. The title 'New Venue' is displayed. Below it are two input fields: 'Name' and 'Capacity'. The 'Name' field is empty. The 'Capacity' field contains the value '234232323'. To the right of the capacity field is a note: 'Capacity must be less than or equal to 500'. Below these fields are dropdown menus for 'Floor' and 'Availability', both set to 'Available'. At the bottom are 'Back' and 'Submit' buttons.

Figure 5.90: Venues - Create Venue Page

This screenshot shows the same 'New Venue' creation page as Figure 5.90, but with validation errors. The 'Name' field is now highlighted in red, indicating it is a required field. A tooltip-like message 'Name is a required field.' appears below the field. The other fields ('Capacity', 'Floor', 'Availability') remain the same as in Figure 5.90.

Figure 5.91: Venues - Create Venue Page With Validation

From the venues page the can also deactivate a venue. Deactivating a resource in the context of the STDC is a soft deletion of that object. A soft deletion is when the resource is deactivated and invisible to users except for the admin. This is done to prevent accidental data loss as well as data integrity. The below are the modals that appear when user tries to deactivate or activate a venue.

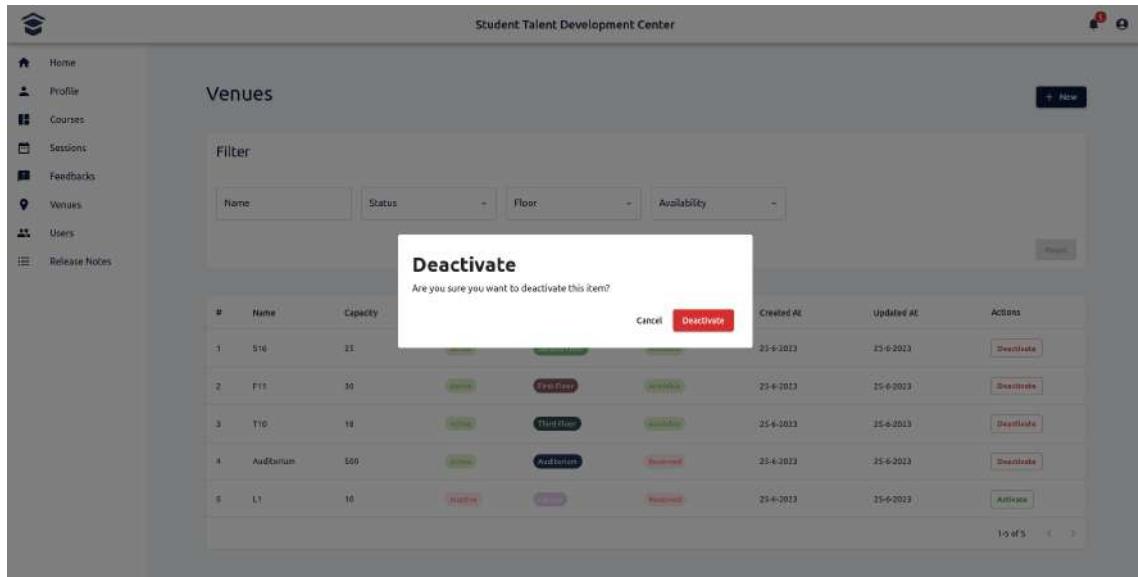


Figure 5.92: Deactivate Modal

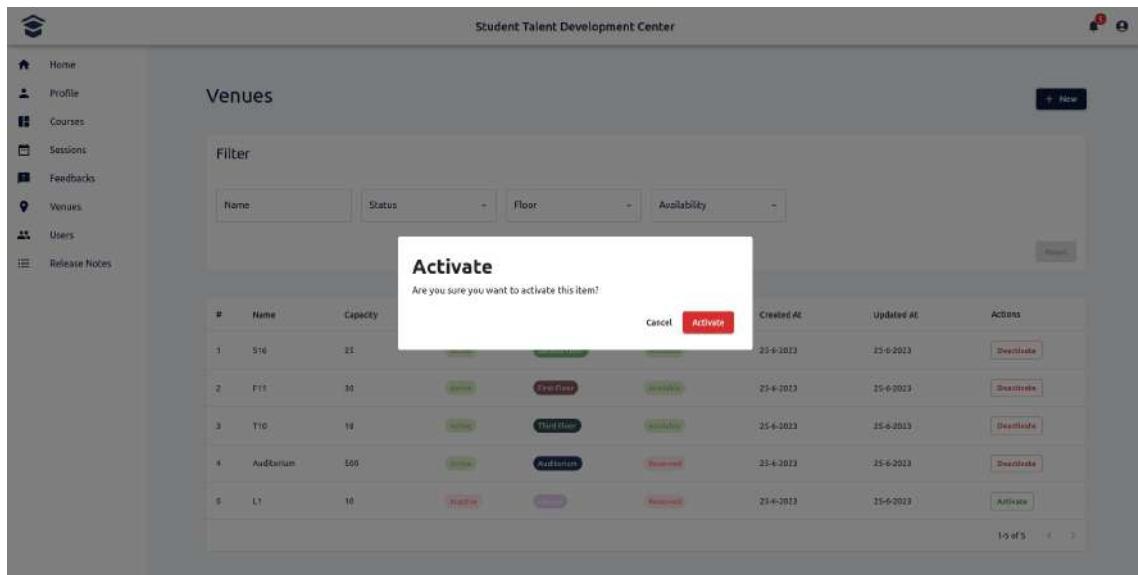


Figure 5.93: Activate Modal

Courses Page

Course page is the page where the superadmin can view all the courses of the application and manage the courses. The below page is the courses table. Note that earlier we focused on courses shown from the home page, the below example is the courses page for superadmin that is managing the course.

#	Name	Status	Tutor	Rating	Created At	Updated At	Actions
1	Tele Communications	Inactive	Wiya Monnet	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Activate</button>
2	Structured Algorithms	Active	Tarik Ahmed	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
3	Automata Theory	Active	Hossein Hosseini	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
4	Embedded Systems	Active	Wiya Monnet	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
5	Artificial Intelligence	Active	Tarik Ahmed	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
6	Robotics	Active	Goward Salihi	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
7	Software Security	Active	Goward Salihi	☆☆☆☆☆	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>

Figure 5.94: Courses Page - Courses Table

From courses' page, the admin can add new courses by clicking the new button. This will redirect to create course page where you can add new courses. The below figures show the create course page.

The screenshot shows the 'New Course' form. It has two input fields: 'Name' and 'User'. Below the fields are 'Back' and 'Submit' buttons. At the bottom of the page, there is a footer with copyright information and a logo.

Figure 5.95: Courses - Create Course Page

Each row of the course table is also clickable and will redirect to show course page where the users can see the sessions of that course as well as the properties of the course including name, tutor, and status. The below figures shows the show course

page and its sub pages.

The screenshot shows the 'Software Testing' course page. On the left is a sidebar with links: Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main content area has a dark header 'Student Talent Development Center' and a title 'Software Testing'. Below it are tabs 'Sessions' and 'Properties'. The 'Sessions' tab is active, displaying four session cards:

- ST - Lecture 04**: Jun 23th - 12:00 PM, Hossein Hassani, T10. Buttons: View Session.
- ST - Lecture 03**: Jun 25th - 1:45 PM, Hossein Hassani, S16. Buttons: View Session.
- ST - Lecture 02**: Jun 25th - 2:00 PM, Hossein Hassani, F11. Buttons: View Session.
- ST - Lecture 01**: Jun 25th - 2:00 PM, Hossein Hassani, T10. Buttons: View Session.

On the right side, there are three notifications: 'Next session on 25th', 'Hossein Hassani', and 'Next session at T10'. A yellow button 'Assign Tutor' is visible. At the bottom, there's a footer with university information and a logo.

Figure 5.96: Courses - Show Course Page

This screenshot shows the 'Properties' tab for the 'Software Testing' course. The layout is identical to Figure 5.96, with the sidebar and header. The 'Properties' tab is active, showing a table with two columns: 'Property' and 'Information'.

Property	Information
Name	Software Testing
Tutor	Hossein Hassani
Rating	4.00 / 5.00
Status	Active

The right side features three notifications and the 'Assign Tutor' button. The footer contains university details and a logo.

Figure 5.97: Courses - Show Course Page - Properties Tab

Assign Tutor button can be seen from both the tables' page and the show page of courses. This page allows for assigning a tutor to a course.

Student Talent Development Center

Courses / Details / Assign Tutor

Assign Tutor to Course

Course: Software Testing User:

Back Submit

University of Mulfidhan Huda (All Rights Reserved © 2023)
Student Talent Development Center
Muhammad Abuzafal

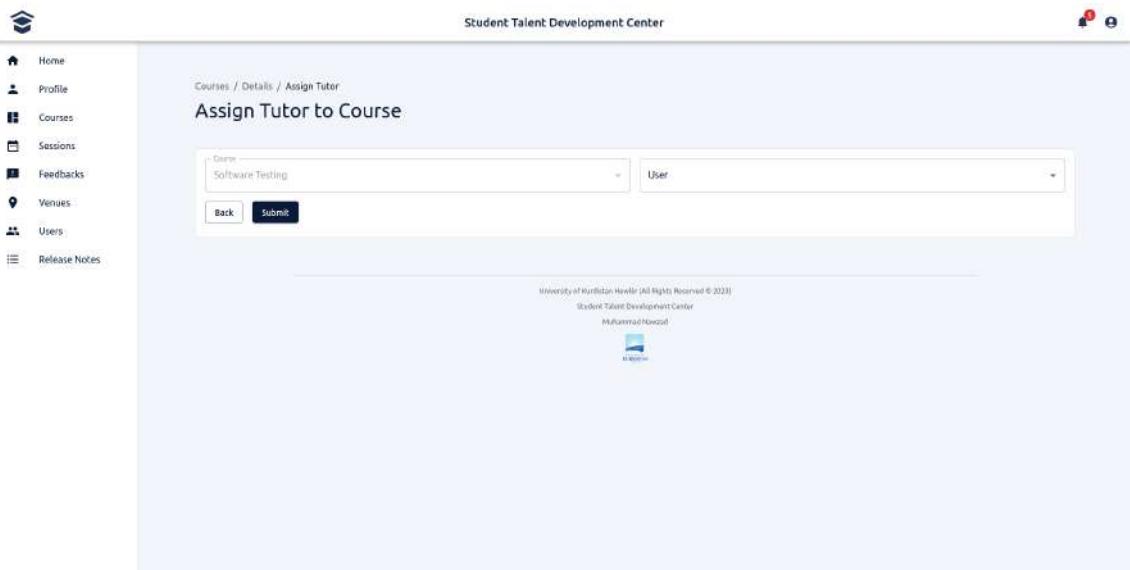



Figure 5.98: Courses - Assign Tutor Page

Feedbacks Page

Feedbacks page is the page where the superadmin can view all the feedbacks of the application and manage the feedbacks. The below page is the feedbacks table. It can be noted that the feedbacks page also make navigation to questions page available, more on this later. Other than questions, you can see that when a feedback is received, first the state of that feedback is unread, when opened it updates to read.

The screenshot shows the 'Feedbacks' section of the 'Student Talent Development Center'. On the left is a sidebar with links: Home, Profile, Courses, Sessions, Feedbacks (which is selected and highlighted in blue), Venues, Users, and Release Notes. The main area has a title 'Feedbacks' and a 'View Questions' button. Below is a 'Filter' section with dropdowns for 'Feedback From', 'Feedback For', and 'State'. A table lists one feedback entry:

#	Feedback From	Feedback For	Rating	State	Created At	Updated At	Actions
1	Muhammad Nawaz	ST - Lecture 01	★★★★★	unread	25-6-2023	25-6-2023	View Feedback

At the bottom right of the table are buttons for '1-1 of 1' and arrows. The footer includes the university's name, copyright information, and a logo.

Figure 5.99: Feedbacks Page - Feedbacks Table

The screenshot shows the 'Feedback' section of the 'Student Talent Development Center'. The sidebar is identical to Figure 5.99. The main area has a title 'Feedback' and a sub-section titled 'Muhammad Nawaz - ST - Lecture 01' with a submission date of 'Submitted on 6/25/2023'. It shows a 5-star rating icon. Below is a comment field: 'Comment: No comment here.' Underneath is a question and answer: 'Q1. How did you like the tutor's teaching style? What is your thought on the tutor's teaching methodology? Did they use any illustrations? Were they engaging?' followed by the response 'Very engaging.' The footer is identical to Figure 5.99.

Figure 5.100: Feedbacks Page - View Feedback

Providing feedback is done by students from sessions page. When the provide feed-

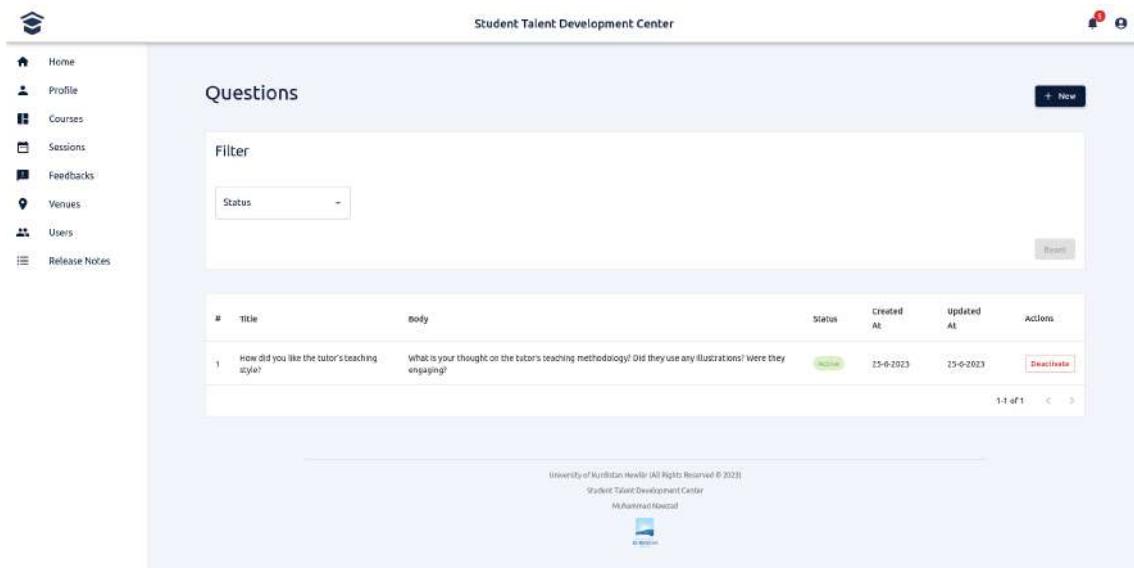
back tab is opened, by default the feedback form asks for a comment and rating, the rest of the form are the currently active questions that are listed as text fields in an intuitive way.

The screenshot shows a web-based application interface for the 'Student Talent Development Center'. On the left, there is a sidebar with icons and labels for Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main content area has a dark header bar with the text 'ST - Lecture 01'. Below the header, there are several tabs: Properties, Attendance, Registered Students, Assign Time and Venue, Provide Feedback, Upload Material, and Materials. The 'Provide Feedback' tab is currently selected. In the center, there is a form titled 'Provide Feedback'. It includes fields for 'Name' (Muhammad Nawaz) and 'Session' (ST - Lecture 01). There is a text area asking 'How did you like the tutor's teaching style?' with the response 'Very engaging.' A 'Comment' section below it says 'No comment here.' At the bottom, there is a rating scale with five stars and the number '4' next to it. Two buttons at the bottom right are labeled 'Cancel' and 'Save'.

Figure 5.101: Provide Feedback from Sessions' Show Page

Questions Page

The Questions Page is almost the same with venues, it is a crud page that allows admin to fully manage them. The importance of questions is that they are dynamically shown to the user for feedback.

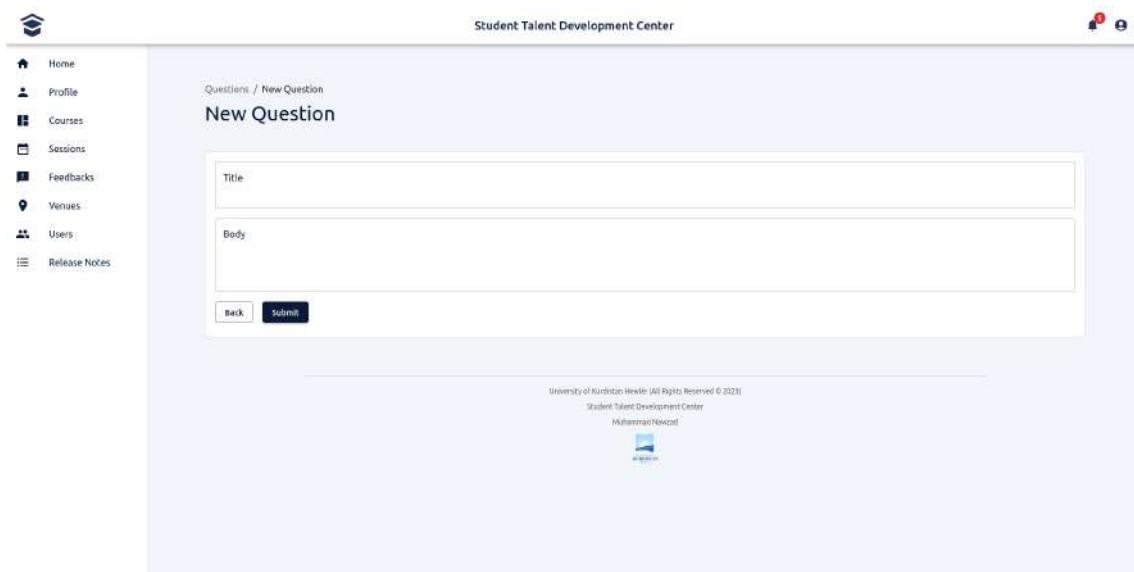


This screenshot shows the 'Questions' page within the 'Student Talent Development Center'. On the left, there's a sidebar with navigation links: Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main area is titled 'Questions' and features a 'Filter' section with a dropdown menu set to 'Status'. A single question is listed in a table:

#	Title	Body	Status	Created At	Updated At	Actions
1	How did you like the tutor's teaching style?	What is your thought on the tutor's teaching methodology? Did they use any illustrations? Were they engaging?	Active	25-6-2023	25-6-2023	<button>Deactivate</button>

At the bottom right of the table, it says '1-1 of 1'. The footer contains copyright information: 'University of Kurdistan Hewlêr (All Rights Reserved © 2023)', 'Student Talent Development Center', 'Muhammad Alawza', and the logo of 'Armenia'.

Figure 5.102: Questions Page - Questions Table



This screenshot shows the 'New Question' page within the 'Student Talent Development Center'. The sidebar on the left is identical to Figure 5.102. The main area is titled 'New Question' and contains two input fields: 'Title' and 'Body'. Below these fields are two buttons: 'Back' and 'Submit'. The footer is identical to Figure 5.102, with the same copyright and logo information.

Figure 5.103: Questions Page - Create Question

Sessions Page

Unlike the other pages, sessions pages are complex. To begin with, viewing all sessions collectively has two perspectives. First is for users that are not admin, they see the card view of sessions. The superadmin on the other hand sees both card view and table view. Changing from a view to another is done through a toggle button that is visible to the admin only.

Name	Date	Time	Tutor	Venue
CA - Lecture 01	Jun 29th	03:00 PM	Wrya Monnet	T18
ES - L01	Jun 27th	05:00 PM	Wrya Monnet	F11
AI - L02	Jul 05th	12:00 PM	Tark Ahmed	F11
AI - L01	Jul 5th	12:00 PM	Tark Ahmed	T10
Security - L01	Jun 26th	12:00 PM	Gowind Selli	T18
Security - L02	Jun 28th	02:25 PM	Gowind Selli	S10
ST - Lecture 04	Jul 5th	12:30 PM	Hosseini Hassani	T10
ST - Lecture 03	Jun 28th	01:45 PM	Hosseini Hassani	S10
ST - Lecture 02	Jun 26th	02:00 PM	Hosseini Hassani	F11
ST - Lecture 01	Jun 25th	02:00 PM	Hosseini Hassani	T10

Figure 5.104: Sessions Page - Card View

Name	Tutor	Status	Date	Start Time	End Time	Rating	Course	Venue	Duration	Cancellation	Created At	Updated At	Actions
CA - Lecture 01	Wrya Monnet	Active	29-6-2023	03:00 PM	06:00 PM	5.0	Computer Architecture	T10	3h 0m	OnGoing	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
ES - L01	Wrya Monnet	Active	27-6-2023	05:00 PM	06:00 PM	5.0	Embedded Systems	F11	1h 0m	OnGoing	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>
AI - L02	Tark Ahmed	Active	6-7-2023	12:00 PM	02:00 PM	5.0	Artificial Intelligence	F11	2h 0m	OnGoing	25-6-2023	25-6-2023	<button>Edit</button> <button>Deactivate</button>

Figure 5.105: Sessions Page - Table View

It is important to note that, again inactive sessions will be visible to admin only just

like the course's. A button to show the history of sessions attended by the current user is also available. When clicked, it will redirect to the show session history page.

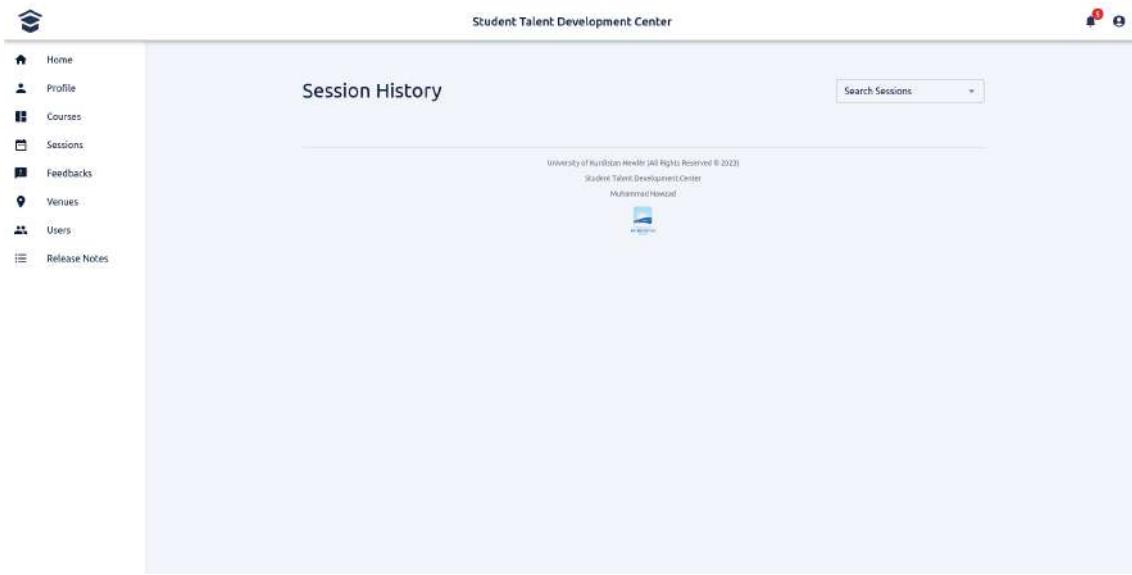


Figure 5.106: Sessions Page - Session History

When the admin decides to create a session, the create session page is shown. The create session page is a complex page that allows the admin to create a session with all the required information. The below figures show the create session page, in this figure, the user tries to put the start time of the session after the end time. This will result in a validation error that stops the user from submitting. Other fields also have validations on them.

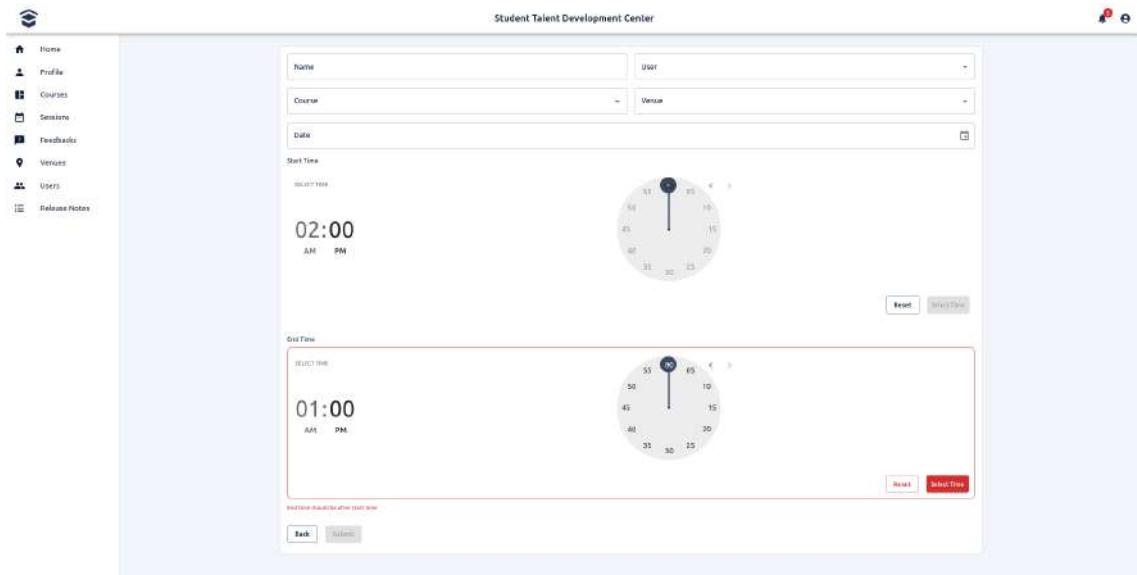


Figure 5.107: Sessions Page - Create Session Page With Validation

Editing a session is also available just like all the other resources mentioned prior to this. However, showing a session is a bit more complex than the others. When a session is viewed, the details of the session is shown in a table and in the banner on top. However the show session page, have many tabs. These tabs are shown or hidden away based on the role of the user.

Properties		Information	
Name	ST - Lecture 01	Tutor	Hossein Hussaini
Date	29-06-2023	Status	Active
Start Time	02:00 PM	End Time	03:15 PM
Rating	★★★★	Venue	T10
Course	Software Testing	Duration	1h 15m
Cancellation	Completed		

Figure 5.108: Sessions Page - Show Session Page

The above shows a session that have been completed. Therefore, cancelation of the

session or registering for the session is not permitted. Below here are a few different perspectives of the show session page. Additionally from viewer's perspective the register button is shown.

Property	Information
Name	AI - L02
Tutor	Tarik Ahmed
Date	06-07-2023
Status	Active
Start Time	12:00 PM
End Time	02:00 PM
Rating	4.5
Venue	F11
Course	Artificial Intelligence
Duration	2h:0m
Cancellation	On going

Figure 5.109: Sessions Page - Show Session Page - Viewer Perspective

Property	Information
Name	AI - L02
Tutor	Tarik Ahmed
Date	06-07-2023
Status	Active
Start Time	12:00 PM
End Time	02:00 PM
Rating	4.5
Venue	F11
Course	Artificial Intelligence
Duration	2h:0m
Cancellation	On going

Figure 5.110: Sessions Page - Show Session Page - Student Perspective

The below figures show each tab of the show session page. First is the properties tab which shows the details of the session in a table. Then is attendance, this tab shows the list of the attendees and allows for entering attendance and removal as well. After that is the registered students. Next is the Assign time and venue tab

which allows the admin to assign a venue and a time for the session if it hasn't been set before. Provide feedback tab is the next tab which we have already covered. The last two tabs are the interesting tabs concerning upload and accessing materials. The upload material tab allows the user to upload a material to the session and allows the tutor to set the material invisible or visible in case of early upload prior to session. The materials tab, allows the users to download the materials uploaded, if the permissions are met, then materials can be hidden or shown, renamed, and deleted.

The screenshot shows the 'Attendance' tab for 'ST - Lecture 01'. The page header includes the title 'Student Talent Development Center' and a sidebar with navigation links: Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main content area displays a table of attendees:

Attendee	Email	Action
Didam Goran	didam.goran@uith.edu.krd	<button>Remove Attendee</button>
Ramin Khalifa	ramin.khalifa@uith.edu.krd	<button>Remove Attendee</button>
Haroon Ahmed	haroon.ahmed@uith.edu.krd	<button>Remove Attendee</button>

At the bottom of the page, there is a footer with copyright information: 'University of Kurdistan Hewler (All Rights Reserved © 2023)', 'Student Talent Development Center', 'Hyderabad House', and the logo of 'al-ameen.uo'.

Figure 5.111: Show Session Page - Attendance Tab

The screenshot shows the 'Registered Students' tab for 'ST - Lecture 01'. The page structure is identical to Figure 5.111, with the 'Attendance' tab being active. The main content area displays a table of registered students:

Student	Email
Didam Goran	didam.goran@uith.edu.krd
Ramin Khalifa	ramin.khalifa@uith.edu.krd
Haroon Ahmed	haroon.ahmed@uith.edu.krd

The footer information is identical to Figure 5.111.

Figure 5.112: Show Session Page - Registered Students Tab

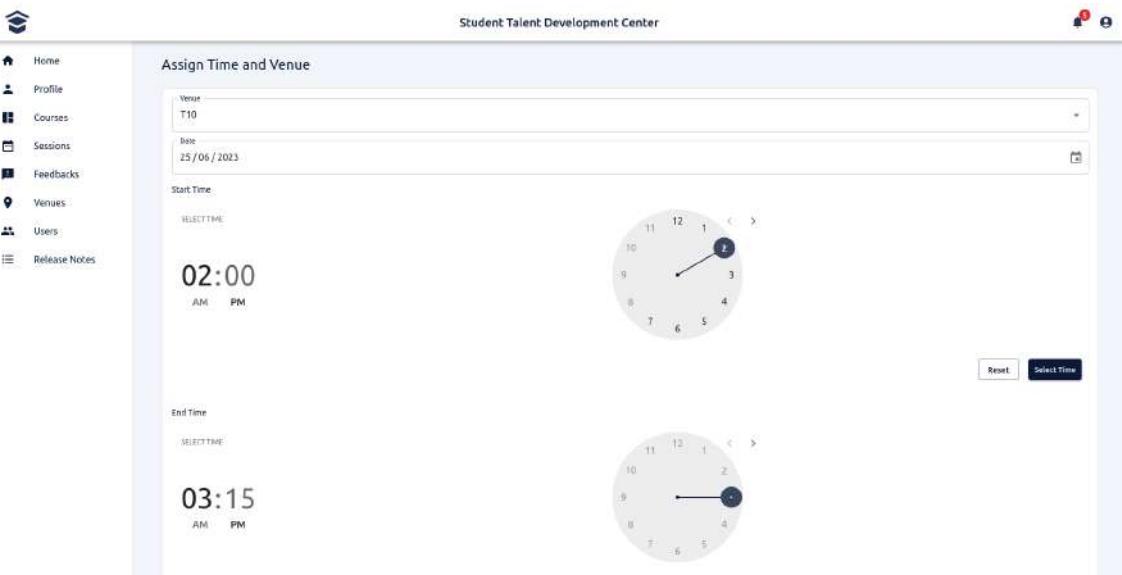


Figure 5.113: Show Session Page - Assign Time and Venue Tab

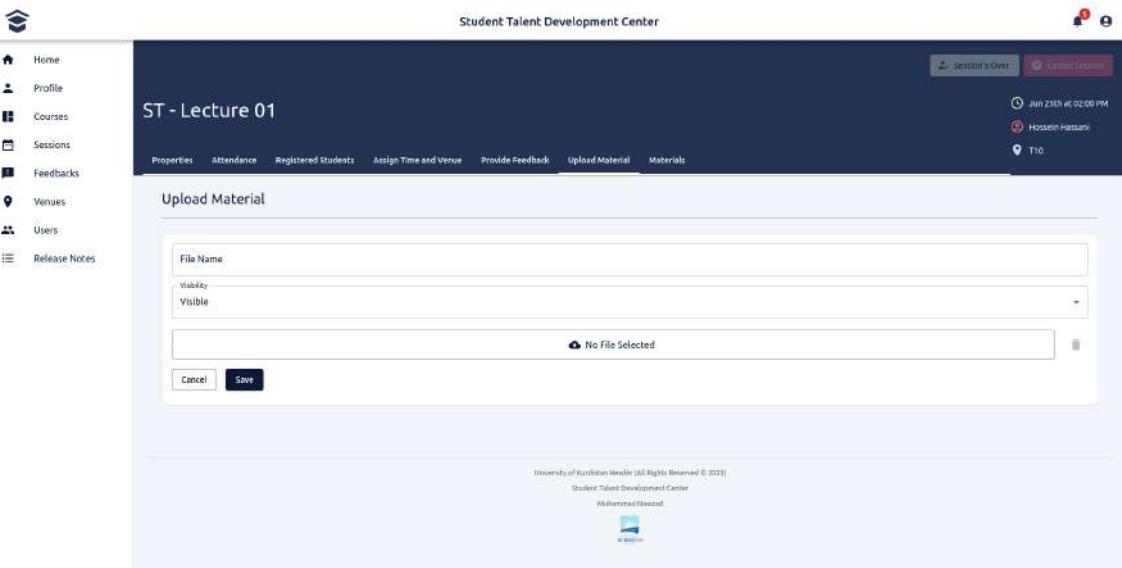


Figure 5.114: Show Session Page - Upload Material Tab

The screenshot shows the 'Student Talent Development Center' interface. On the left, a sidebar lists navigation options: Home, Profile, Courses, Sessions, Feedbacks, Venues, Users, and Release Notes. The main content area is titled 'ST - Lecture 01'. At the top right, there are buttons for 'Session is Over', 'Cancel session', 'JUN 23th at 02:00 PM', 'Hossein Hassan', and 'T10'. Below the title, a navigation bar includes 'Properties', 'Attendance', 'Registered Students', 'Assign Time and Venue', 'Provide Feedback', 'Upload Material', and 'Materials'. The 'Materials' tab is selected. Under the 'Materials' heading, there is a file named 'Lecture 01 - Introduction.pdf' (532 KB). Below this, under 'Invisible Materials', there is a file named 'Lecture 02 - What is Testing.pdf' (application/pdf). A context menu on the right side of the page offers options: Download, Rename, Show, and Delete.

Figure 5.115: Show Session Page - Materials Tab

Other Pages

The other pages are the pages that are not directly related to the resources of the application or they are static pages. They are as follows.

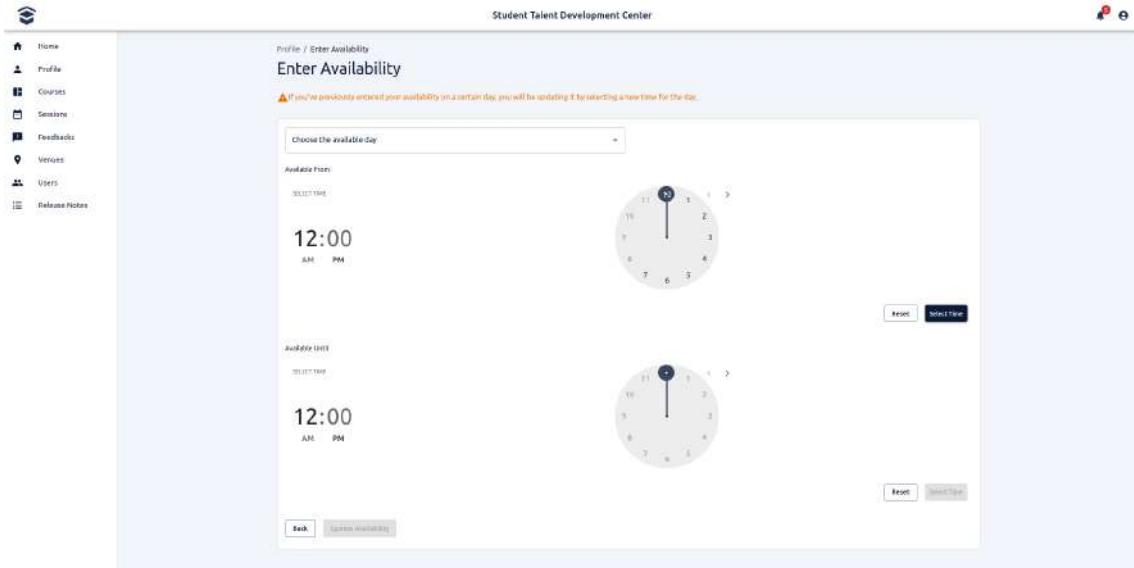


Figure 5.116: Enter Availability Page

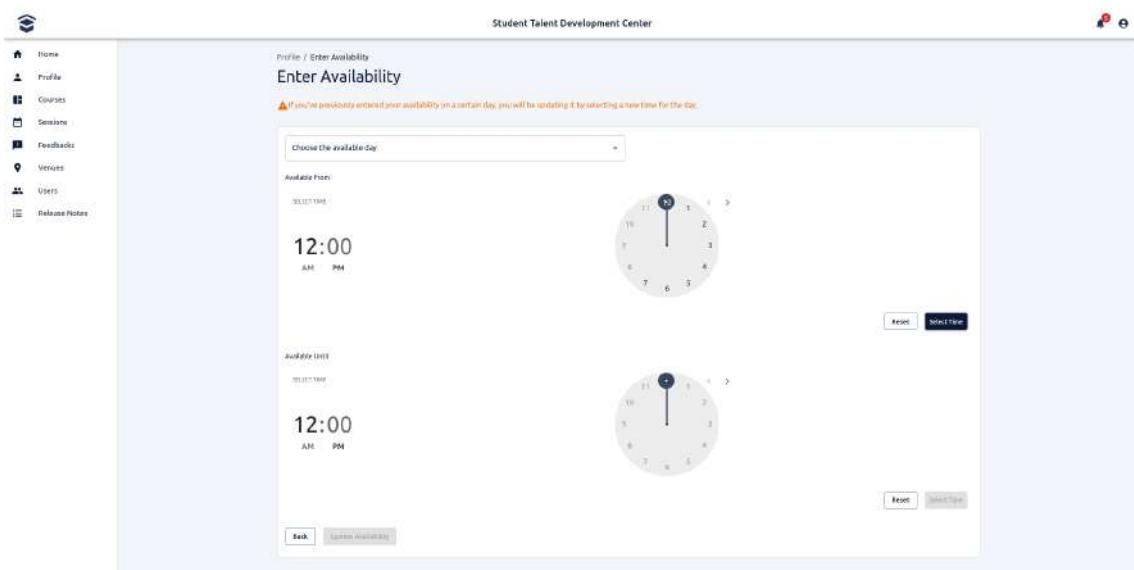


Figure 5.117: Enter Availability Page

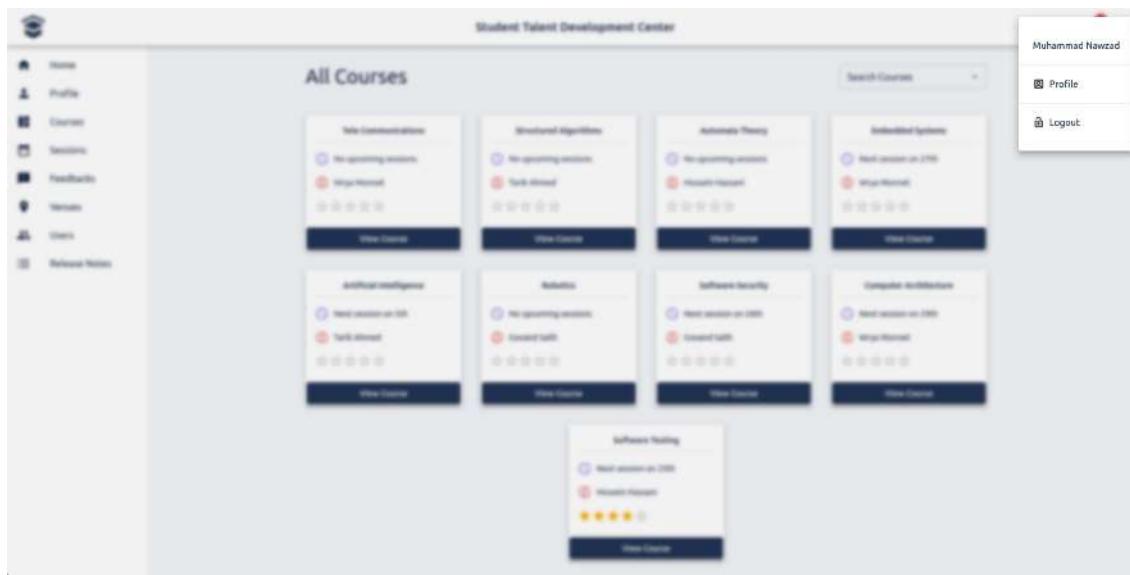


Figure 5.118: Logout Button

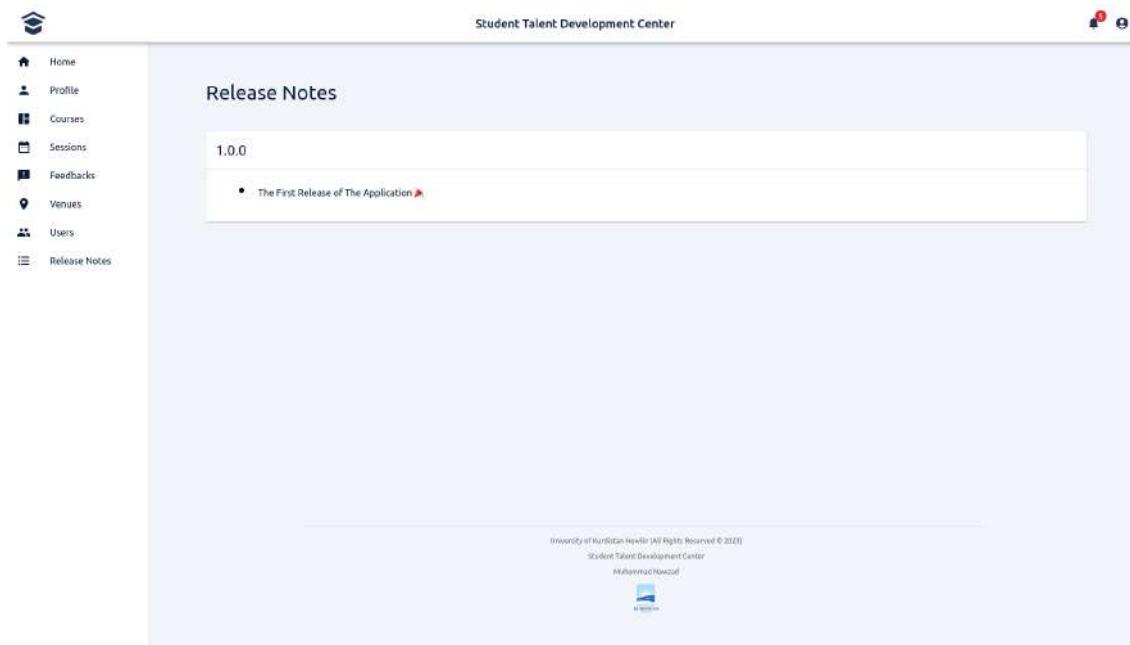


Figure 5.119: Release Notes Page

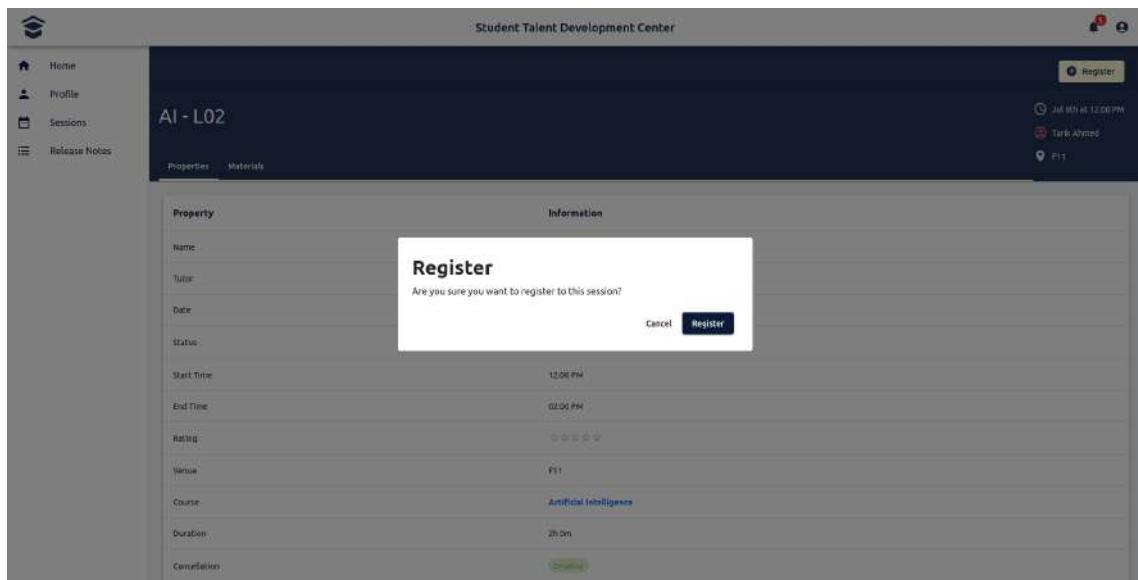


Figure 5.120: Register Modal

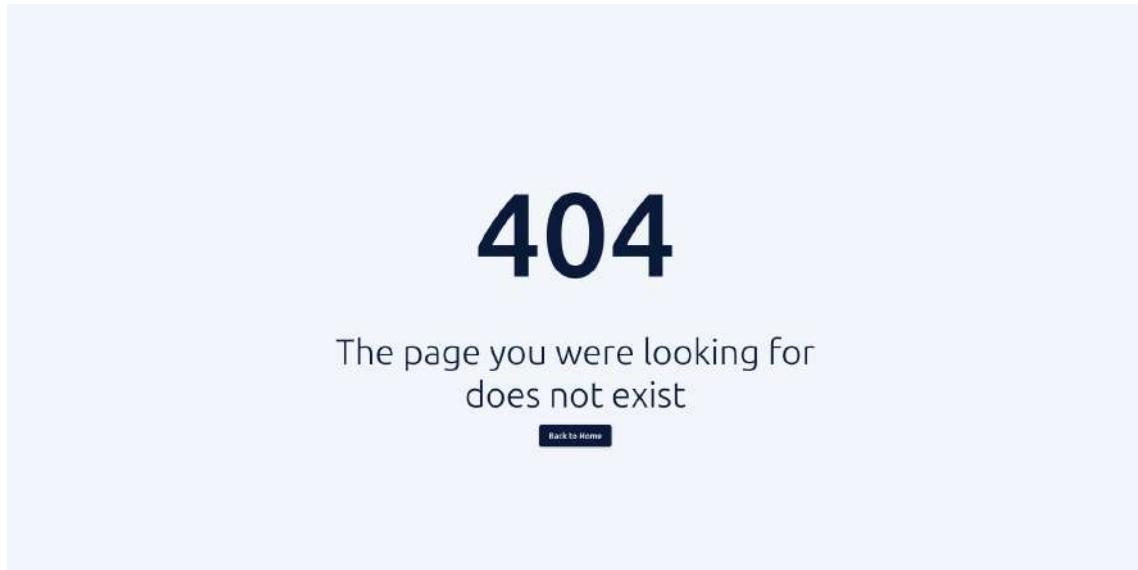


Figure 5.121: 404 Not Found Page

403

Access to the requested
resource is forbidden

[Back to Home](#)

Figure 5.122: 403 Forbidden Page

Summary

In conclusion, despite the fact that the STDC Application is currently capable of performing all of its intended functions, there are still a number of areas in which it could be enhanced, both in terms of its overall performance and the experience it provides its users. The application has the potential to attract more students and effectively communicate the mission and services of the STDC if these improvements are implemented.

To begin, the inclusion of a modern portal that captivates potential learners and presents the STDC as an appealing and cutting-edge option for meeting their educational requirements would be a significant improvement to the application and would yield significant benefits. This portal has the potential to highlight the many different programs and opportunities that are offered, ultimately motivating students to become a part of the STDC.

It is recommended to include dedicated sections such as "About Us," "Contact Us," and a page dedicated to frequently asked questions in order to provide users with comprehensive information. These additions will provide users with a more in-depth understanding of the objectives of the STDC, make communication with the organization simpler, and provide effective responses to frequently asked questions.

It is absolutely necessary, in order to raise levels of user satisfaction, to work on the application's user interface (UI). The application can be made easier to use and more interesting for its users by incorporating the user-friendly functionalities that were suggested earlier, such as visually appealing design elements, streamlined processes, and intuitive navigation. In this way, the application will become more intuitive.

Another essential component that requires immediate attention is the quality of the source code. Documentation of the existing codebase should be carried out in great detail, and work is needed to be done to both reduce redundant code and improve the structure of the code. This will not only improve the application's maintainability and scalability, but it will also make it simpler for future developers to understand the

code and make changes to it.

Because the Windows Live login will no longer be supported in the future, it is strongly recommended that user authentication be moved to Azure AD. Users who access the application will now have access to a login mechanism that is both secure and dependable as a result of this change, which ensures compatibility with the most recent technology standards.

In conclusion, the incorporation of notifications into the application has the potential to significantly boost user engagement. By putting in place a notification system, the STDC will be able to keep users promptly informed about any updates, changes, or important announcements that are related to the STDC center. This will ensure that students remain informed and connected with the organization.

5.2 Testing

The following section will contain the Testing Phase of the Software Development Life Cycle of the Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System.

5.2.1 Test Plan

This part will outline the test plan for the web application project for the Student Talent Development Center. The scope, methods, tools, and timing of the anticipated test activities that will be used to evaluate this application are all specified in this document. Furthermore, the test plan serves as a blueprint to conduct software testing activities as a defined process.

Test Scope

The project's testing scope will solely concentrate on components of the system itself. Used services that are used in the project but whose logic is not visible to the system will not be tested, however only their integration within the system will be tested. Users logging in is one example of this. The system expects an access token obtained using OAuth 2.0 and OpenID Connect from Microsoft Accounts Services. The system's back-end does not care how the access token is received as long as it is a legitimate access token, therefore despite the fact that login logic happens in the Microsoft Accounts Services, it is wholly external to the system. However, the system will test the internal validation of the obtained access token. Concerns about the system's internal components should be tested. External system-related concerns won't be considered during testing.

Test Approach

The testing phase of the Software Development Life Cycle of the Student Talent Development Center: Design and Implementation of Scheduling and Resources Management Web Application System, will be divided into two sections, The Automated Testing Techniques and The Manual Testing Techniques. Several sub-techniques will be used which include the followings.

- Automated Testing
 - Unit Testing
 - Integration Testing
 - Build Verification Testing
- Manual Testing
 - Requirement Specifications Testing
 - Model Based Testing
 - Boundary Analysis Testing
 - Inspection Testing
 - Usability Testing
 - User Acceptance Testing

Test Environment

There are three distinct environments that will be used in the development of the project. The environments are as follows:

1. Discovery
2. Development
3. Production

All of the environments will be used for testing.

Test Tools

The project will make use of a number of notable technologies, including:

1. Lucid Chart: Used for model testing and state transitions.
2. Faker: Used for fake data generation for automated testing.
3. RSpec: Used for writing automated testing.

4. Shoulda Matchers: Used for providing RSpec-compatible one-liners for testing.
5. Heroku's Console: Used for testing the system in the production environment, mainly for Inspection Testing.

Test Schedule

In this project, testing starts as soon as requirement collecting is done. The requirements will be used to create test cases, which will be maintained as the process progresses. The project's analysis and design phases will include some of the manual testing methods. When development begins, automated testing will begin. Despite the fact that testing begins in the analysis phase and lasts until the project is handed over to the client, testing is an integral part of the development process from the very beginning. Because the same test case may be run several times, it is important to be able to track changes to the test case over time, thus requirements traceability matrix is used.

Test Cases

The test cases for this project can be found the Appendices (A46). Each test case shall be traced through out the project in accordance to the next section (Requirements Traceability Matrix).

5.2.2 Requirements Traceability Matrix

A Requirements Traceability Matrix (RTM) is a document or tool used to track the mapping between the requirements specified for a software system and the tests that have been written to validate those requirements. The RTM serves as a link between the requirements and the test cases, ensuring that all requirements are tested and that all test cases are traceable back to specific requirements. This helps in better traceability and accountability of the requirements, reduces the risk of missing any requirements during testing, and improves overall quality of the software. The Requirements Traceability Matrix for this project is shown in the figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7.

Table 5.1: Requirements Traceability Matrix (RTM) - 1

No.	Associated ID	Test Case ID	Environment	UML Ref	Module Ref (Code)	Document Ref	Date Revised	UI Ref	Status
1	RQ01	TC01	Discovery & Development	UC01, UCM2	spec/requests/v1/ user_spec.rb	FSM & RST	22 June 2023	UI01	Passed
2	RQ02	TC02	Discovery & Development	UC02, UCM2	spec/requests/v1/ attendance_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
3	RQ03	TC03	Discovery & Development	UC03, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI04	Passed
4	RQ04	TC04	Discovery & Development	UC04, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI09	Passed
5	RQ05	TC05	Discovery & Development	UC05, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI04	Passed
6	RQ06	TC06	Discovery & Development	UC06, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI04	Passed
7	RQ07	TC07	Discovery & Development	UC07, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI04	Passed
8	RQ08	TC08	Discovery & Development	UC08, UCM2	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI05	Passed
9	RQ09	TC09	Discovery & Development	UC09, UCM2, AM1	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
10	RQ10	TC10	Discovery & Development	UC10, UCM2	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI05	Passed
11	RQ11	TC11	Discovery & Development	UC11, UCM2	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI05	Passed
12	RQ12	TC12	Discovery & Development	UC12, UCM2	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI05	Passed
13	RQ13	TC13	Discovery & Development	UC13, UCM2	spec/requests/v1/ venue_spec.rb	Integration Test & FSM & RST	22 June 2023	UI07	Passed
14	RQ14	TC14	Discovery & Development	UC14, UCM2	spec/requests/v1/ venue_spec.rb	Integration Test & FSM & RST	22 June 2023	UI07	Passed
15	RQ15	TC15	Discovery & Development	UC15, UCM2	spec/requests/v1/ venue_spec.rb	Integration Test & FSM & RST	22 June 2023	UI07	Passed
16	RQ16	TC16	Discovery & Development	UC16, UCM2	spec/requests/v1/ venue_spec.rb	Integration Test & FSM & RST	22 June 2023	UI07	Passed
17	RQ17	TC17	Discovery & Development	UC17, UCM2	spec/requests/v1/ venue_spec.rb	Integration Test & FSM & RST	22 June 2023	UI07	Passed

Table 5.2: Requirements Traceability Matrix (RTM) - 2

18	RQ18	TC18	Discovery & Development	UC18, UCM2	spec/requests/v1/ user_spec.rb	Integration Test & FSM & RST	22 June 2023	UI08	Passed
19	RQ19	TC19	Discovery & Development	UC19, UCM2	spec/requests/v1/ course_spec.rb	Integration Test & FSM & RST	22 June 2023	UI04	Passed
20	RQ20	TC20	Discovery & Development	UC20, UCM2	spec/requests/v1/ attendance_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
21	RQ21	TC21	Discovery & Development	UC21, UCM3	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
22	RQ22	TC22	Discovery & Development	UC22, UCM3	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
23	RQ23	TC23	Discovery & Development	UC23, UCM3	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI05	Passed
24	RQ24	TC24	Discovery & Development	UC24, UCM3	spec/requests/v1/ user_spec.rb	Integration Test & FSM & RST	22 June 2023	UI03	Passed
25	RQ25	TC25	Discovery & Development	UC25, UCM3	spec/requests/v1/ session_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
26	RQ26	TC26	Discovery & Development	UC26, UCM3	spec/requests/v1/ user_spec.rb	Integration Test & FSM & RST	22 June 2023	UI03	Passed
27	RQ27	TC27	Discovery & Development	UC27, UCM4, AM2	spec/requests/v1/ feedback_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
28	RQ28	TC28	Discovery & Development	UC28, UCM4	spec/requests/v1/ feedback_spec.rb	Integration Test & FSM & RST	22 June 2023	UI06	Passed
29	RQ29	TC29	Discovery & Development	UC29, UCM5	spec/requests/v1/ material_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
30	RQ30	TC30	Discovery & Development	UC30, UCM5, AM3	spec/requests/v1/ material_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
31	RQ31	TC31	Discovery & Development	UC31, UM4	spec/requests/v1/ feedback_spec.rb	Integration Test & FSM & RST	22 June 2023	UI06	Passed
32	RQ32	TC32	Discovery & Development	UC32, UM2	spec/requests/v1/ user_spec.rb	Integration Test & FSM & RST	22 June 2023	UI03	Passed
33	RQ33	TC33	Discovery & Development	UC33, UM3	spec/requests/v1/ attendance_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
34	RQ34	TC34	Discovery & Development	UC34, UM3	spec/requests/v1/ attendance_spec.rb	Integration Test & FSM & RST	22 June 2023	UI10	Passed
35	RQ35	TC35	Discovery & Development	UC35, UM4	spec/requests/v1/ question_spec.rb	Integration Test & RST	22 June 2023	-	Passed

Table 5.3: Requirements Traceability Matrix (RTM) - 3

36	RQ36	TC36	Discovery & Development	UC36, UJM4	spec/requests/v1/question_spec.rb	Integration Test & RST	22 June 2023	-	Passed
37	RQ37	TC37	Discovery & Development	UC37, UJM4	spec/requests/v1/question_spec.rb	Integration Test & RST	22 June 2023	UI11	Passed
38	RQ38	TC38	Discovery & Development	UC38, UJM4	spec/requests/v1/question_spec.rb	Integration Test & RST	22 June 2023	-	Passed
39	RQ27	TC39	Development	UC27	spec/models/answer_spec.rb	Unit Testing	22 June 2023	UI11	Passed
40	RQ27	TC40	Development	UC27	spec/models/answer_spec.rb	Unit Testing	22 June 2023	UI11	Passed
41	RQ27	TC41	Development	UC27	spec/models/answer_spec.rb	Unit Testing	22 June 2023	UI11	Passed
42	RQ27	TC42	Development	UC27	spec/models/answer_spec.rb	Unit Testing	22 June 2023	UI11	Passed
43	RQ27	TC43	Development	UC27	spec/models/answer_spec.rb	Unit Testing	22 June 2023	UI11	Passed
44	RQ02	TC44	Development	UC02	spec/models/attendance_spec.rb	Unit Testing	22 June 2023	UI11	Passed
45	RQ02	TC45	Development	UC02	spec/models/attendance_spec.rb	Unit Testing	22 June 2023	UI11	Passed
46	RQ02	TC46	Development	UC02	spec/models/attendance_spec.rb	Unit Testing	22 June 2023	UI11	Passed
47	RQ02	TC47	Development	UC02	spec/models/attendance_spec.rb	Unit Testing	22 June 2023	UI11	Passed
48	RQ02	TC48	Development	UC02	spec/models/attendance_spec.rb	Unit Testing	22 June 2023	UI11	Passed
49	RQ02	TC49	Development	UC02	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed
50	RQ05	TC50	Development	UC05	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed
51	RQ05	TC51	Development	UC05	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed
52	RQ05	TC52	Development	UC05	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed
53	RQ05	TC53	Development	UC05	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed

Table 5.4: Requirements Traceability Matrix (RTM) - 4

54	RQ05	TC54	Development	UC05	spec/models/course_spec.rb	Unit Testing	22 June 2023	UI11	Passed
55	RQ04	TC55	Development	UC04	spec/models/enrollment_spec.rb	Unit Testing	22 June 2023	UI11	Passed
56	RQ04	TC56	Development	UC04	spec/models/enrollment_spec.rb	Unit Testing	22 June 2023	UI11	Passed
57	RQ04	TC57	Development	UC04	spec/models/enrollment_spec.rb	Unit Testing	22 June 2023	UI11	Passed
58	RQ27	TC58	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
59	RQ27	TC59	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
60	RQ27	TC60	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
61	RQ27	TC61	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
62	RQ27	TC62	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
63	RQ27	TC63	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
64	RQ27	TC64	Development	UC27	spec/models/specification_spec.rb	Unit Testing	22 June 2023	UI11	Passed
65	RQ37	TC65	Development	UC37	spec/models/question_spec.rb	Unit Testing	22 June 2023	UI11	Passed
66	RQ37	TC66	Development	UC37	spec/models/question_spec.rb	Unit Testing	22 June 2023	UI11	Passed
67	RQ37	TC67	Development	UC37	spec/models/question_spec.rb	Unit Testing	22 June 2023	UI11	Passed
68	RQ37	TC68	Development	UC37	spec/models/question_spec.rb	Unit Testing	22 June 2023	UI11	Passed
69	RQ37	TC69	Development	UC37	spec/models/question_spec.rb	Unit Testing	22 June 2023	UI11	Passed
70	RQ30	TC70	Development	UC30	spec/models/material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
71	RQ30	TC71	Development	UC30	spec/models/material_spec.rb	Unit Testing	22 June 2023	UI11	Passed

Table 5.5: Requirements Traceability Matrix (RTM) - 5

72	RQ30	TC72	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
73	RQ30	TC73	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
74	RQ30	TC74	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
75	RQ30	TC75	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
76	RQ30	TC76	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
77	RQ30	TC77	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
78	RQ30	TC78	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
79	RQ30	TC79	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
80	RQ30	TC80	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
81	RQ30	TC81	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
82	RQ30	TC82	Development	UC30	spec/models/ material_spec.rb	Unit Testing	22 June 2023	UI11	Passed
83	RQ15	TC83	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
84	RQ15	TC84	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
85	RQ15	TC85	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
86	RQ15	TC86	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
87	RQ15	TC87	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
88	RQ15	TC88	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed
89	RQ15	TC89	Development	UC15	spec/models/ venue_spec.rb	Unit Testing	22 June 2023	UI11	Passed

Table 5.6: Requirements Traceability Matrix (RTM) - 6

90	RQ01	TC90	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
91	RQ01	TC91	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
92	RQ01	TC92	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
93	RQ01	TC93	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
94	RQ01	TC94	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
95	RQ01	TC95	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
96	RQ01	TC96	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
97	RQ01	TC97	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
98	RQ01	TC98	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
99	RQ01	TC99	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
100	RQ01	TC100	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
101	RQ01	TC101	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
102	RQ01	TC102	Development	UC01	spec/models/ user_spec.rb	Unit Testing	22 June 2023	UI11	Passed
103	RQ10	TC103	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
104	RQ10	TC104	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
105	RQ10	TC105	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
106	RQ10	TC106	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
107	RQ10	TC107	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed

Table 5.7: Requirements Traceability Matrix (RTM) - 7

108	RQ10	TC108	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
109	RQ10	TC109	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
110	RQ10	TC110	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
111	RQ10	TC111	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
112	RQ10	TC112	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
113	RQ10	TC113	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed
114	RQ10	TC114	Development	UC10	spec/models/ session_spec.rb	Unit Testing	22 June 2023	UI11	Passed

5.2.3 Requirement Specifications Testing

In the requirements specification testing for the software being developed, a thorough evaluation of the requirements has been conducted, categorizing them by components and users. The feedback has been used to improve the requirements in order to ensure that the software meets the needs of its users and stakeholders. The feedback can be found in the Appendices (A43).

In the "System Core" component, it was initially required that all users must be able to login, however, it was suggested that user must use UKH email and password to ensure that only authorized users are able to access the system. Additionally, it was specified that all users must see how many modules are taught in STDC and that users should be able to view them as a list. Furthermore, it was specified that students and tutors should see how many modules they are enrolled in and that the user sees enrolled modules first when listing modules.

For the Admin user, it was initially specified that they must be able to manage modules and sessions, however this needed to be changed to be more specific and to be broken down such that Admin must be able to view, create, update, and delete modules and sessions, additionally, it was suggested that they also do the same for venues which was missing from the requirements and that each task be separated into different requirements. Moreover, it was specified that the Admin should be able to see the list of attendance of each session, and when viewing a session, there must be a button for this purpose.

In the Scheduling component, it was specified that all users must see upcoming lesson's time and venue, which will be shown when viewing the session. Furthermore, for the Admin user, it was specified that they could see how many hours each tutor has tutored, which will be shown when viewing their profile.

In the Feedback component, it was specified that students should be able to provide feedback after each session, and there should be a button to indicate "Provide Feedback".

In the Resource component, it was specified that students should access the material

uploaded for each module grouped by sessions, and files (Materials) are uploaded to session, not module.

It was noted that some requirements were repetitions of previous ones, such as “view hours tutored”, and some requirements were very generalized such as “manage module and sessions”. It was suggested to keep requirements consistent, be more specific with requirements (breakdown manage to show, list, create, update, and delete), change the component name “Basic functionalities” to “System Core”, and rewrite the requirements in a different format that doesn’t have sub-requirements.

Overall, the requirements have been evaluated and improved to ensure that the software meets the needs of its users and stakeholders, and that it is fit for its intended purpose.

5.2.4 Model Based Testing

Here an approximate Finite State Machine (FSM) has been constructed based on the system's user interface design as illustrated in the figure 4.9 and a state transition table as illustrated in table 4.9 which revealed that the system's user interface design does not adhere to specifications and poses significant usability obstacles.

The FSM indicates various unsatisfactory state transitions includes the followings:

- When admin views feedbacks page the "View Feedback" button which views an individual feedback is not specified in the requirements and has not been accommodated for.
- The requirement for users to be able to view their profile details is missing from the initial requirements and has not been accommodated for however available in the UI.
- The requirement for students to register for a session is missing from the initial requirements and has not been accommodated for however seen in the UI.
- The requirement for tutors to view the list of students in a scheduled session is missing from the initial requirements and has not been accommodated however available in the UI.

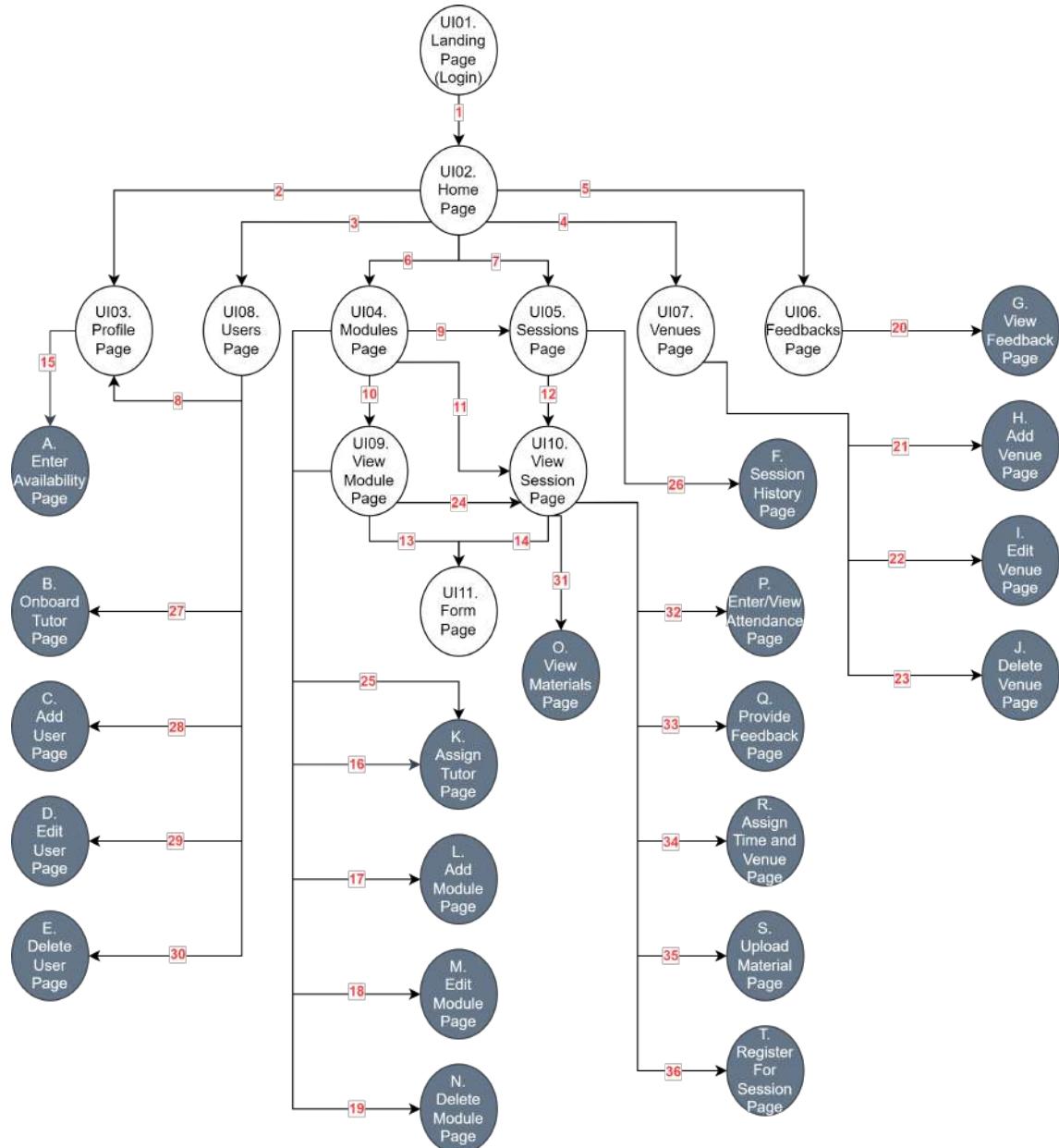


Figure 5.123: FSM State Transitions

The rows correspond to current states, while the columns indicate next states. The row/column intersections indicate actions associated with the state transitions. There are cases in which multiple different actions result in the same transition. The actions are enumerated and shown in Table 5.9 presentation reasons. The N/A is an automated action that leads back to the UI11. Form Page since there are a lot of them, it is not shown in the diagram for simplicity. It is also worth mentioning that the A,B..T pages doesn't not have a specific UI designed for them in the previous chapter. However, they are known user interfaces that are form alike. The numbers on top are abbreviations like this UI01 => 1, UI07 => 7, and so on.

Table 5.9: Actions and their references

Action	Reference
1	Login Button Click
2	Profile Button Click from Sidebar
3	Users Button Click from Sidebar
4	Modules Button Click from Sidebar
5	Sessions Button Click from Sidebar
6	Venues Button Click from Sidebar
7	Feedbacks Button Click from Sidebar
8	Click on User's Row in Users Page
9	View Sessions Button Click
10	Click on Module's Row from Modules Page
11	View Session Button Click from Module Page
12	View Session Button Click from Sessions Page
13	Assign Tutor Button Click from Module Page
14	Click on Session's tab buttons
15	Enter Availability Button from Profile Page
16	Assign Tutor Button Click from Modules Page
17	Add Button Click from Module Page
18	Edit Button Click from Module Page
19	Delete Button Click from Module Page
20	View Feedback Button from Feedbacks Page
21	Add Button Click from Venue Page
22	Edit Button Click from Venue Page
23	Delete Button Click from Venue Page
24	View Session Button Click from Module Page
25	Assign Tutor Button Click from Module Page
26	View Session History Button from Sessions Page
27	Onboard Tutor Button from Users Page
28	Add Button Click from User Page
29	Edit Button Click from User Page
30	Delete Button Click from User Page
31	Materials Tab Click from Session Page
32	Attendance Tab Click from Session Page
33	Provide Feedback Tab Click from Session Page
34	Assign Time and Venue Tab Click from Session Page
35	Upload Material Tab Click from Session Page
36	Register Tab Click from Session Page
N/A	Not Available (Automated action that redirected to Form Page)

5.2.5 Boundary Analysis Testing

In boundary analysis testing, inputs that define the 'boundaries' of a component's functionality are examined. These limits include minimum and maximum limits as well as all permitted values within that range. It is necessary to conduct boundary testing because errors tend to occur at the input domain's limits. In the context of the STDC, boundary analysis testing was performed on TextFields and any other user-input fields.

Throughout the testing of the STDC application, various input types demonstrated consistent behavior. These input categories were evaluated:

1. **Auto Complete Text Field:** These fields enable the user to browse for predetermined options. The user cannot select options that are not included in the list, thus encouraging data precision. It was observed that all Auto Complete Text Fields in the application exhibited consistent behavior.
2. **Text Field:** This is one of the most frequently utilized STDC application components. Although the component functioned as anticipated, it lacked obvious boundary restrictions. The examined limits included:
 - Empty Field: Users are not allowed to submit an empty field and are informed when a required field is left blank.
 - Symbols: The field accepts all types of symbols.
 - Integers: All integer inputs are accepted.
 - Characters: The field accepts all characters.
 - Allowed Length: There appeared to be no limit to the length of input, with the application accepting even 10,000-character-long course names. Thus it could result in the below undesired user interfaces.

#	Name
1	LYKwjkRPLzaNqBqWsbkVOuTzn7MqzVUV2kgSUmDwMUqeXy9CA2NgDTRUxIqJIIxOm2m3ou2Vbn2dvb6v9KVF4F7Rdkwjny@hv6xgwyfhn@UdsUvwrexZtsf4HZBxYeIUcGCB9gau/TmMLc33cnITNWrlObj,O1A95cYocWZmbd/J5Cbdl
2	dbcnj329yrr94yrfnlkjndskjfnfdskjnfksndkjkjnskjdnFkjnsnf93u24u04302840328048230482038402fkndsnkfrnsfkndsfkjnskfjkjnsnf3u034034200324
3	Object Oriented
4	Software Testing

1-4 of 4 < >

Figure 5.124: Invalid course name in courses table



Figure 5.125: Invalid course name in course profile

3. **File Upload:** This feature, available in the Resource Management System, allows users to upload any type of file, one at a time. A potential improvement could be to allow multiple file uploads simultaneously, which would enhance user experience and efficiency.
4. **Selection:** This component functioned well, only allowing options within the predefined boundary to be selected.
5. **Rating:** Users are permitted to select whole numbers between 1 and 5, with the system correctly prohibiting the selection of negative numbers or any numbers outside of the given range.
6. **Date Picker:** Date pickers functioned well, disallowing the selection of past dates for future sessions and the selection of weekends. However, the system didn't prevent the selection of holidays, which could be a potential area of improvement.

7. **Time Picker:** These functioned efficiently, restricting time selection outside of the UKH's working hours (8 AM to 8 PM) and ensuring that the start time of a session wasn't later than the end time.
8. **Numeric Text Field:** These were used exclusively in venue creation. The behavior of these fields was unusual:

- Character Input: As expected, character input wasn't allowed.
- Capacity Input: Users were able to enter a capacity of 0, a negative number, or even a number as high as 9999999999. This behavior is quite irregular and could be problematic as the typical capacity for a UKH venue ranges from 10 to 500.

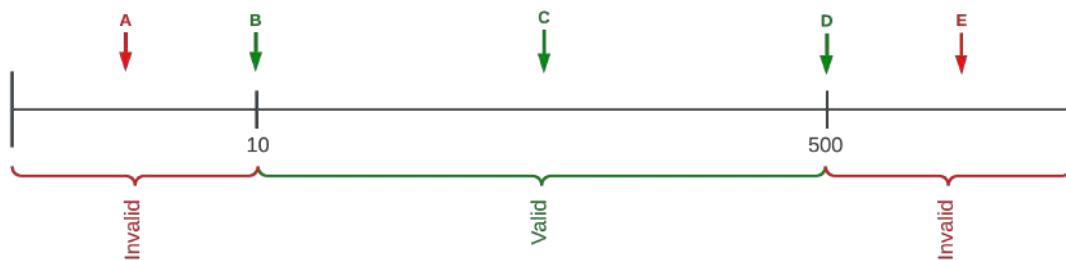


Figure 5.126: Numeric boundary analysis

Table 5.10: Numeric boundary analysis - test cases' table

Test Case	Result
A	Invalid
B	Valid
C	Valid
D	Valid
E	Invalid

#	Name	Capacity	Status
1	gteagfdgfd	0	Active
2	dsfdsfsd	999999999	Active
3	dsdsdsd	-999999999	Active
4	ewewweew	-99999999	Active
5	rwerre	-1	Active

Figure 5.127: Created venues with invalid capacities

The results of boundary analysis testing for the STDC app showed how crucial it is to establish limits and boundaries in order to guarantee reliable data, enhance the user experience, and prevent bugs. Several potential enhancements were noted, including:

- One way to prevent users from entering unnecessarily long data is to enforce a maximum length for input in Text Fields.
- Adding the ability to upload several files at once to the File Upload function.
- Including minimum and maximum capacity validations in the Numeric Text Fields used for venue construction.
- To prevent the selection of holidays, a new validation should be added to the Date Picker.

As a result of resolving these findings and implementing the proposed enhancements, the STDC program will become more efficient and user-friendly. It highlights the value of boundary analysis testing in spotting vulnerabilities and making a program more stable.

5.2.6 Inspection Testing

During the course of an inspection test of the Student Talent Development Center (STDC) program, a number of major concerns were discovered, and the objective of this section is to offer a detailed explanation and evaluation of those issues. Because software reliability is essential to the usability and durability of a software application, such careful inspection tests are essential to find, explain, and then fix the many faults that could hinder the user experience. This is because software reliability is key to the usability and longevity of a software application.

In this section, we are going to concentrate on a manual inspection test, which entails doing a comprehensive investigation into the functioning of the system in order to locate any possible flaws or mistakes.

The STDC program was seen engaging in erroneous behavior on two separate occasions, both of which interfered with the software's capacity to perform its typical operations as described below. In the next section, we will examine each of these situations one at a time, providing specifics about the problem that arose, the investigation method that was utilized, the identification of the flaw, and the recommended solution.

Case One: Tutor Login Issue

The lack of ability of the application to authenticate a user who was attempting to log in with the role of "Tutor" was the first problem that was found during the manual inspection test. The authentication of users is the most important part of their interaction with the system; thus, this was a very serious issue.

When an attempt was made to log in as a Tutor, the system refused access. Further research using the network part of the browser's developer tool revealed that the "current-user" endpoint on the server had returned an error code of 401. This HTTP status code is typically sent in situations when authentication was requested but either failed or was not yet given. The fact that this error was generated suggested that the software was experiencing difficulties validating the user's login credentials.

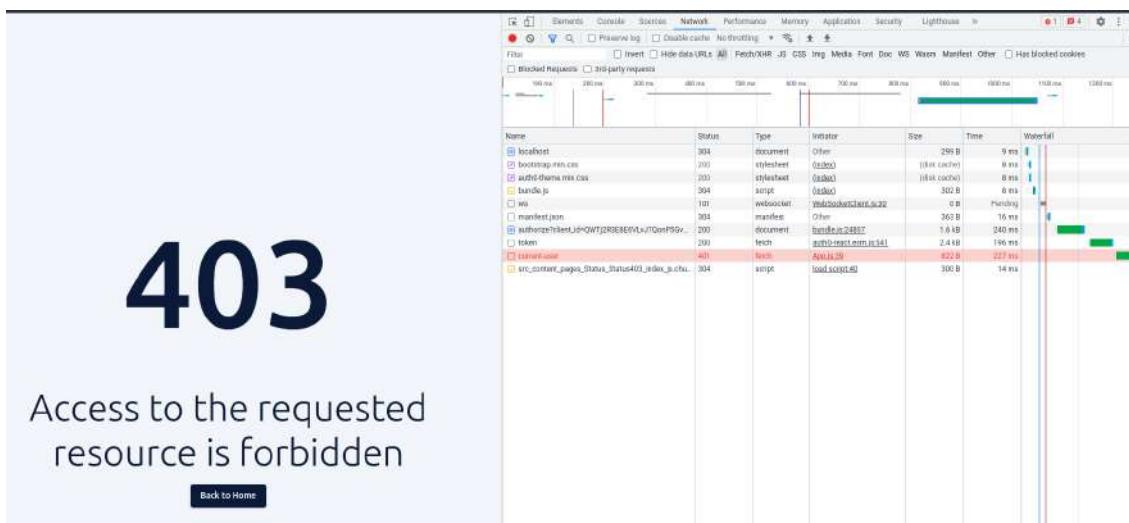


Figure 5.128: Inspection testing - HTTP 403 when login

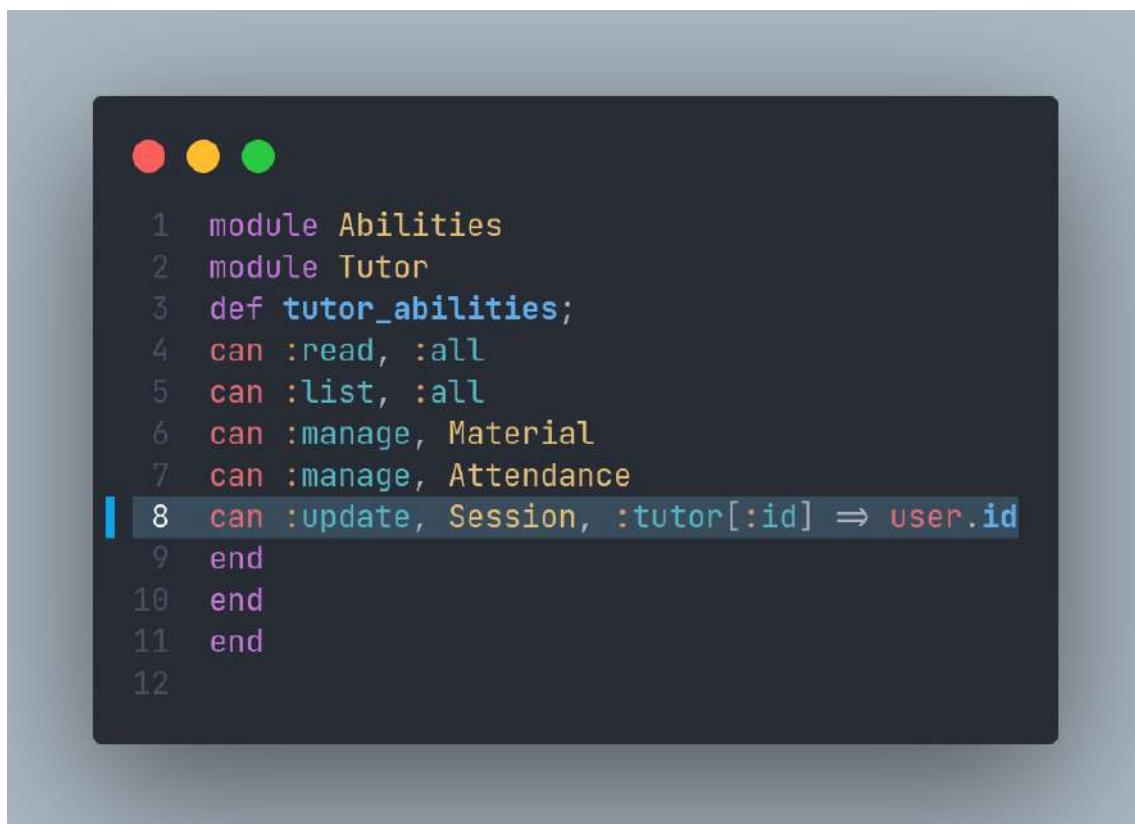
In order to figure out what was causing this problem in the first place, the database that was hosted on the server had to be erased. This choice was made on the basis of the idea that an incorrect user object in the database may possibly be the cause of the issue. This step, however, inevitably led to another problem: the "Superadmin" user, who is immediately seeded into the database upon deployment and carries both the "Superadmin" and "Tutor" roles, was also unable to log in to the system. This user is automatically seeded into the database upon deployment.

Because the client was simply given a 401 Unauthorized error, it was determined that this problem should be investigated locally using the localhost. Copying the token that was used to log in as the superadmin user and using it for local testing was done so that the circumstances on the client side could be replicated.

During the course of the investigation, it was discovered that the error's stack trace led to the "ability.rb" model located in the models directory. The STDC application's user permissions are handled by this model, which is responsible for handling those permissions. Notably, if this file experiences an exception, it will send a 401 HTTP response by default. This explains both the server's 403 error and the client's 401 error when they try to retrieve current user information from the server.

Following more investigation, the "tutor_abilities" method was found to contain the problematic piece of code that was responsible for throwing the error. To be more specific, the program code attempted to retrieve the tutor ID from the tutor field of the user's record (which was a JSONB field in the database), compare it with the user's ID, but it failed to do so because of a syntax issue and instead raised an exception.

The incorrect line of code was: `tutor[:id]: user.id`, which indicated that the program was making an effort to get the id key from the tutor JSONB field and evaluate it in comparison to the `user.id`. Despite this, `:tutor[:id]` was the source of an error since it had improper syntax. In addition, the user object wasn't created yet at that point in the code, which means that the `user.id` variable was also the source of the issue.

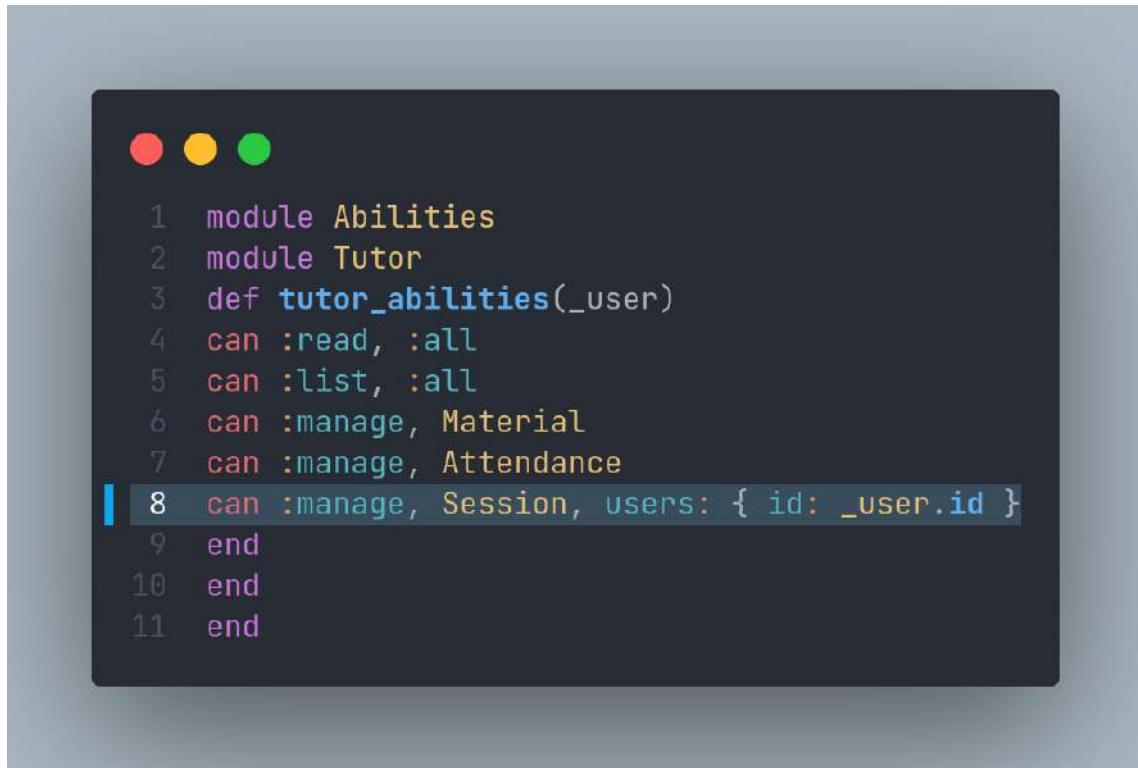


A screenshot of a terminal window showing a Ruby script. The script defines a module `Abilities` with a nested module `Tutor`. It contains several `can` statements for various abilities like `:read`, `:list`, `:manage` for `Material` and `Attendance`. The problematic line is highlighted with a blue selection bar: `can :update, Session, :tutor[:id] => user.id`. This line is invalid because it attempts to access the `:id` key from the `:tutor` field, which is a JSONB field and cannot be directly compared with `user.id`.

```
1 module Abilities
2 module Tutor
3   def tutor_abilities;
4     can :read, :all
5     can :list, :all
6     can :manage, Material
7     can :manage, Attendance
8     can :update, Session, :tutor[:id] => user.id
9   end
10  end
11 end
12
```

Figure 5.129: Inspection testing - Case One Defected Line

The following is the implementation of the solution to correct this problem which is illustrated in the figure that can be seen below.



```
1 module Abilities
2 module Tutor
3   def tutor_abilities(_user)
4     can :read, :all
5     can :list, :all
6     can :manage, Material
7     can :manage, Attendance
8     can :manage, Session, users: { id: _user.id }
9   end
10 end
11 end
```

Figure 5.130: Inspection testing - Case One Solution

Case Two: Session Search Issue

The lack of ability of the application to authenticate a user who was attempting to log in with the role of "Tutor" was the first problem that was found during the manual inspection test. The authentication of users is the most important part of their interaction with the system; thus, this was a very serious issue.

The second issue that we discovered during our inspection test relates to a problem with the search capability that can be found on the Sessions page. Within the STDC program, one of the most important functions is performed by the Sessions page, which displays all of the enrolled and readily available sessions. Users are given the ability to locate certain sessions in a timely manner thanks to the inclusion of a search bar on this page. However, during testing it was determined that this feature had a fundamental problem, which undermines the usability that was intended for the website.

When a user tries to find an existing session, the autocomplete function offers suit-

able session options depending on the keywords that have been typed. When a user clicks on a session from the search results, the functionality of this feature is supposed to take them to the page of the selected session. This is the intended behavior. Nevertheless, when testing, it was noticed that selecting a session result does not lead the user to the appropriate location when they click on it. Instead, it essentially fills the autocomplete search bar with the name of the selected session, keeping the user rooted on the Sessions page throughout the process.

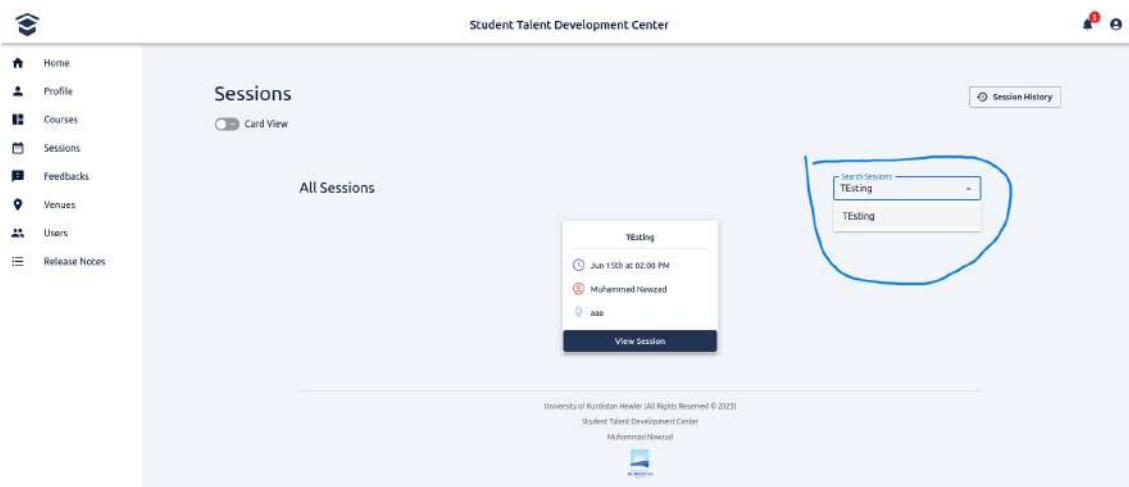
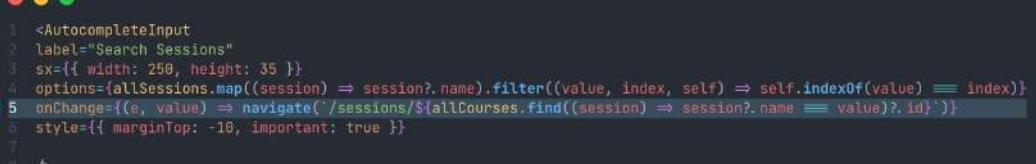


Figure 5.131: Inspection testing - Case Two defective search

This confusing problem called for a more thorough investigation of the onClick event that is connected to the search results. When a user clicks on a session name from the search results, this event is responsible for the action that is executed as a result of that click. It was thought that a mistake occurring during this moment may be the root cause of the observed dysfunction.

After doing an investigation into the code, it was found that the onClick event's method had been supplying inaccurate information on the session's redirect link. Due to the fact that this link was broken, the system was unable to properly redirect the user to the session page that they intended. The 'allSessions' useState, which is supposed to store all of the session information, was specifically built so that the link generating technique could access the session ID from there. On the other hand, because of a mistake in the coding, it was attempting to access the 'allCourses' useState instead, which is not declared in the file that is relevant to the discussion.



```
1 <AutocompleteInput
2   label="Search Sessions"
3   sx={{ width: 250, height: 35 }}
4   options={allSessions.map((session) => session?.name).filter((value, index, self) => self.indexOf(value) === index)}
5   onChange={(e, value) => navigate('/sessions/${allCourses.find((session) => session?.name === value)?.id}'})
6   style={{ marginTop: -10, important: true }}
7
8 />
```

Figure 5.132: Inspection testing - Case Two defected line

The answer to this situation was not overly complicated to figure out. The problem might be fixed by substituting the undefined 'allCourses' useState with the specified 'allSessions' useState instead. After making this adjustment, the functionality of the search box restored to how it should have been behaving. After the user makes their selection from the list of search results, they are immediately sent to the page that corresponds to the selected session.

The solution to this problem is illustrated in the figure below.



```
1 <AutocompleteInput
2   label="Search Sessions"
3   sx={{ width: 250, height: 35 }}
4   options={allSessions.map((session) => session?.name).filter((value, index, self) => self.indexOf(value) === index)}
5   onChange={(e, value) => navigate('/sessions/${allSessions.find((session) => session?.name === value)?.id}'})
6   style={{ marginTop: -10, important: true }}
7
8 />
```

Figure 5.133: Inspection testing - Case Two Solution

Summary

During the course of the manual inspection test, we found two major issues with the STDC application. The need of thorough testing in the field of software development is made clear by both of these examples. Problems with the login process or the search bar can have a significant impact on how the program works for its users. By carefully examining these instances, we may not only pinpoint the issues but also discover their causes, leading to a deeper comprehension of the system as a whole.

First, we looked at a serious vulnerability that was preventing a 'Tutor' user from signing in. After discovering this problem, we dug into the server's responses, inspected the database, and studied the error stack trace in great detail. We tracked down the cause of the login problem to a syntax error in the 'tutor_abilities' function of the 'ability.rb' model. The error was caused by a call to a non-existent object, 'user.id', which was intended to compare the value of the 'id' field in the JSONB tutor object to a value outside the function's scope. By fixing this problem, we were able to bring back a crucial part of the app's operation, login.

The second scenario required us to investigate a malfunctioning search feature on the 'Sessions' page. While the search bar is a vital navigational aid, it was not properly taking users to the appropriate session page. We discovered the session's redirect URL was incorrectly produced during our onClick investigation because the software was trying to retrieve data from an undefined useState. A straightforward yet possibly irritating oversight was uncovered by this investigation. We were able to get the search bar working again after a fast patch of changing the incorrect useState with the proper one.

The complexity and interdependence of the many parts of a software program are shown by these two examples. A single mistake in the code might have a domino effect, resulting in serious problems that affect the program as a whole. This highlights the significance of thorough testing and inspection. By thoroughly testing the system, we can guarantee that the final product will be dependable, effective, and simple to use.

5.2.7 Granularity of Automated Testing

Automated testing plays a crucial role in the development and deployment of high-quality, dependable software, reducing the likelihood of unanticipated behavior or defects once the software reaches end-users. In this section, we will examine the Student Talent Development Center's (STDC) automated testing procedures.

The STDC application consists of four primary components: the Core System, the Feedback System, the Resource Management System, and the Scheduling System. The Core System component is at the core of the application, comprising fundamental features such as user management. The Feedback System functions as a mechanism for collecting user feedback, thereby facilitating the continuous refinement and enhancement of the user experience. The Resource Management System facilitates course and session content management and uploading. Lastly, the Scheduling System assists in coordinating and managing the scheduling of various events and duties within the application.

Granularity in the context of automated testing refers to the level of specificity or detail at which testing is conducted. Granularity within the STDC application has been established at two distinct levels: the method level for unit testing and the class level for integration testing.

Unit testing is a testing strategy that entails testing individual software components, in this instance the class methods. The purpose of unit testing is to isolate a section of code and validate its correctness, enabling developers to detect flaws early in the development cycle and repair them more easily than if they were found later. Unit tests for STDC were written at the method level, concentrating on the simplest testable software component. This procedure ensures that each method, or mini-routine, in the software operates as expected.

Unit testing for the STDC application included a comprehensive examination of validations, enums, and associations. Using Rails as the framework and RSpec as the testing tool, unit testing confirmed that the methods adhered to the specified rules or 'validations', ensuring that the software would not accept invalid data. It ensured

that 'enums', constants with a fixed set of predefined values, functioned accurately by testing them. The 'associations', or relationships between distinct classes in the software, were also examined to validate that the intended interactions between classes occurred. This method-level granularity of unit testing facilitated an inter-class testing scenario, permitting a thorough examination of the interactions between classes.

Integration testing, however, is a higher level than unit testing. It is a method of testing that verifies the compatibility of distinct software components when combined. In the STDC application, integration testing was conducted at the class level, with a focus on the interaction between classes within a component.

This level of granularity demands an in-depth understanding of how classes interact, given that a class may contain numerous methods. The integration tests for STDC focused mainly on testing all of the controller endpoints, or the numerous 'entry points' through which users interface with the application.

This form of integration testing also involved component-to-component interaction. Components, in terms of STDC's architecture, are collections of classes that deliver specialized system functionality. As classes within these components communicated with one another during testing, the application demonstrated inter-component communication. This allowed the identification and correction of any problems that could have arisen when classes from different components interacted, ensuring seamless coordination within the STDC application.

In conclusion, the STDC application's implementation of automated testing has been carefully planned with a keen focus on granularity. Method-level granularity in unit testing allowed for a detailed, micro-level perspective, facilitating the identification and resolution of issues within the application's methods. Class-level granularity in integration testing provided a macro-level perspective and ensured the seamless interaction and integration of various components. Consequently, by leveraging this multi-level granularity in automated testing, the STDC application demonstrates a robust, dependable software development strategy, preserving the quality and efficiency that are essential in today's rapidly evolving technology landscape.

5.2.8 Unit Testing

Unit testing is a crucial aspect of testing of the Student Talent Development Center (STDC), offering a mechanism to validate the correctness and functionality of individual units within the application. Since STDC's backend uses the Ruby on Rails framework, these units are represented by the models, which are responsible for data management, business logic execution, and relationship mapping with other objects. Here, we will explore the unit testing strategies applied to our key models. The testing is facilitated by the RSpec testing framework, enhanced by the Shoulda Matchers library, which aids in creating clean, readable, and effective test cases.

Common Testing Aspects

Both attribute validations and model associations are performed on every model. In order to guarantee that the model's data attributes are correct, attribute validations are performed. On the other hand, model associations are used to set up and confirm the connections between the various models in the system. In addition to associations and associations, in each model creation of an instance of that model is tested by using the subject which will contain an instance of the model if the build of that model is complete.

User Model

The unit tests that have been completed for the User model are as follows.

Attribute Validations



```
1 describe 'Validations' do
2   it { should define_enum_for(:status).with_values(inactive: 0, active: 1) }
3   it { should validate_presence_of(:remote_id) }
4   it { should validate_uniqueness_of(:remote_id) }
5   it { should validate_presence_of(:email) }
6   it { should validate_uniqueness_of(:email) }
7   it { should validate_presence_of(:first_name) }
8   it { should validate_presence_of(:last_name) }
9   it { should validate_presence_of(:name) }
10 end
```

Figure 5.134: User Model - Validation Unit Tests

Presence validation: The validate_presence_of tests confirm that certain attributes must contain values. For example, validate_presence_of(:email) verifies that a User instance cannot be saved without an email attribute.

Uniqueness validation: These tests validate that the value of certain attributes is unique across all instances of a model, ensuring that no two users can share the same value of such an attribute. For instance, validate_uniqueness_of(:email) asserts that each User has a unique email.

Enumeration validation: We use define_enum_for to validate enumerations, where a set of named values correspond to integer values in the database. Our test, define_enum_for(:status).with_values(inactive: 0, active: 1), confirms that the status attribute accurately maps the terms inactive and active to 0 and 1, respectively.

Model Associations

A screenshot of a terminal window with a dark background. At the top left are three colored dots: red, yellow, and green. Below them is a block of RSpec test code:

```
1 describe 'Associations' do
2   it { should have_many(:enrollments) }
3   it { should have_many(:courses).through(:enrollments) }
4   it { should have_many(:attendances) }
5   it { should have_many(:sessions).through(:attendances) }
6 end
```

Figure 5.135: User Model - Association Unit Tests

One-to-many relationships: The have_many test verifies that one instance of the User model can be associated with multiple instances of another model. In our test suite, it should have_many(:enrollments) and it should have_many(:attendances) confirm that a User can have multiple Enrollments and Attendances, respectively.

Many-to-many relationships: The have_many...through test validates relationships where a User instance can be associated with many instances of another model through a third model. In our suite, it should have_many(:courses).through(:enrollme-

nts) and it should have _many(:sessions).through(:attendances) asserts that a User can be associated with many Courses through Enrollments, and with many Sessions through Attendances.



```
1 require 'rails_helper'
2
3 RSpec.describe User, type: :model do
4   subject { build(:user) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should define_enum_for(:status).with_values(inactive: 0, active: 1) }
12    it { should validate_presence_of(:remote_id) }
13    it { should validate_uniqueness_of(:remote_id) }
14    it { should validate_presence_of(:email) }
15    it { should validate_uniqueness_of(:email) }
16    it { should validate_presence_of(:first_name) }
17    it { should validate_presence_of(:last_name) }
18    it { should validate_presence_of(:name) }
19  end
20
21  describe 'Associations' do
22    it { should have_many(:enrollments) }
23    it { should have_many(:courses).through(:enrollments) }
24    it { should have_many(:attendances) }
25    it { should have_many(:sessions).through(:attendances) }
26  end
27 end
28
```

Figure 5.136: User Model Unit Tests

Session Model

The unit tests that have been completed for the Session model are as follows.

Attribute Validations



```
1 describe 'Validations' do
2   it { should validate_presence_of(:name) }
3   it { should validate_presence_of(:start_time) }
4   it { should validate_presence_of(:end_time) }
5   it { should validate_presence_of(:date) }
6   it { should validate_presence_of(:duration) }
7   it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
8   it { should define_enum_for(:cancellation).with_values(ongoing: 0, cancelled: 1, rescheduled: 2, completed: 3) }
9 end
```

Figure 5.137: Session Model - Validation Unit Tests

Presence validation: The validate_presence_of tests are used to ensure that a Session instance cannot be saved without certain critical attributes. This is demonstrated in our suite by tests such as validate_presence_of(:name), validate_presence_of(:start_time), validate_presence_of(:end_time), validate_presence_of(:date), and validate_presence_of(:duration), which ensure that each session must have these key attributes.

Enumeration validation: We use define_enum_for to validate that named values of an attribute accurately correspond to integer values in the database. In our suite, define_enum_for(:status).with_values(active: 0, inactive: 1) verifies the status attribute maps 'active' and 'inactive' to 0 and 1, respectively. define_enum_for(:cancellation).with_values(ongoing: 0, cancelled: 1, rescheduled: 2, completed: 3) ensures that the 'cancellation' attribute maps 'ongoing', 'cancelled', 'rescheduled', and 'completed' to 0, 1, 2, and 3, respectively.

Model Associations



```
1 describe 'Associations' do
2   it { should belong_to(:course)}
3   it { should belong_to(:venue) }
4   it { should have_many(:attendances) }
5   it { should have_many(:users).through(:attendances) }
6 end
```

Figure 5.138: Session Model - Association Unit Tests

One-to-one relationships: The belong_to test checks the presence of one-to-one associations between models. In our suite, it should belong_to(:course) and it should belong_to(:venue) validate that each session belongs to a single course and a single venue.

One-to-many relationships: The have_many test validates that one instance of the Session model can be associated with multiple instances of another model. For instance, it should have_many(:attendances) asserts that a Session can have multiple Attendances.

Many-to-many relationships: The have_many...through test is used when a Session instance can be associated with many instances of another model via a third model. it should have_many(:users).through(:attendances) in our test suite validates that a Session can be associated with many Users through Attendances.



```
require 'rails_helper'

RSpec.describe Session, type: :model do
  subject { build(:session) }

  it 'is valid with valid attributes' do
    expect(subject).to be_valid
  end

  describe 'Validations' do
    it { should validate_presence_of(:name) }
    it { should validate_presence_of(:start_time) }
    it { should validate_presence_of(:end_time) }
    it { should validate_presence_of(:date) }
    it { should validate_presence_of(:duration) }
    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
    it { should define_enum_for(:cancellation).with_values(ongoing: 0, cancelled: 1, rescheduled: 2, completed: 3) }
  end

  describe 'Associations' do
    it { should belong_to(:course) }
    it { should belong_to(:venue) }
    it { should have_many(:attendances) }
    it { should have_many(:users).through(:attendances) }
  end
end
```

Figure 5.139: Session Model Unit Tests

Venue Model

The unit tests that have been completed for the Venue model are as follows.

Attribute Validations



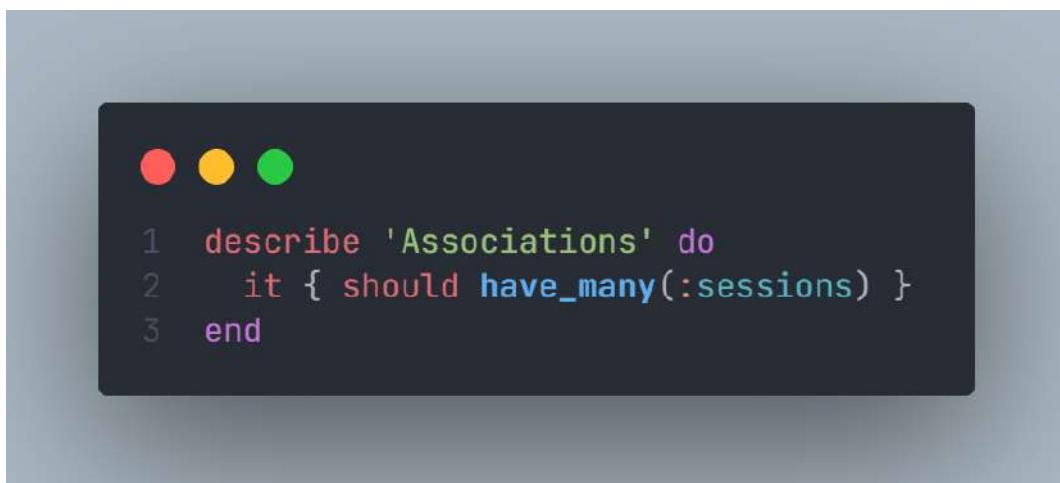
```
describe 'Validations' do
  it { should validate_presence_of(:name) }
  it { should validate_presence_of(:capacity) }
  it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
  it { should define_enum_for(:floor).with_values(ground_floor: 0, first_floor: 1, second_floor: 2, third_floor: 3, library: 4, staff_cafeteria: 5, auditorium: 6, parking_lot: 7) }
  it { should define_enum_for(:availability).with_values(available: 0, reserved: 1) }
end
```

Figure 5.140: Venue Model - Validation Unit Tests

Presence validation: The validate_presence_of tests are designed to confirm that a Venue instance cannot be saved without certain vital attributes. This is depicted in our suite by tests such as validate_presence_of(:name) and validate_presence_of(:capacity), which guarantee that each venue must have a name and a capacity.

Enumeration validation: define_enum_for is used to ensure that named values of an attribute accurately correspond to integer values in the database. In our suite, define_enum_for(:status).with_values(active: 0, inactive: 1) ensures that the status attribute correctly maps 'active' and 'inactive' to 0 and 1, respectively. define_enum_for(:floor).with_values(ground_floor: 0, first_floor: 1, second_floor: 2, third_floor: 3, library: 4, staff_cafeteria: 5, auditorium: 6, parking_lot: 7) confirms that the 'floor' attribute maps correctly to these respective integer values. Moreover, define_enum_for(:availability).with_values(available: 0, reserved: 1) verifies that the availability attribute accurately maps 'available' and 'reserved' to 0 and 1, respectively.

Model Associations

A screenshot of a terminal window with a dark background. At the top left are three colored dots: red, yellow, and green. Below them is some RSpec test code:

```
1 describe 'Associations' do
2   it { should have_many(:sessions) }
3 end
```

Figure 5.141: Venue Model - Association Unit Tests

One-to-many relationships: The have_many test confirms the presence of one-to-many associations between models. In our suite, it should have_many(:sessions) validates that a Venue can host multiple Sessions.



```
require 'rails_helper'

RSpec.describe Course, type: :model do
  subject { build(:venue) }

  it 'is valid with valid attributes' do
    expect(subject).to be_valid
  end

  describe 'Validations' do
    it { should validate_presence_of(:name) }
    it { should validate_presence_of(:capacity) }
    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
    it { should define_enum_for(:floors).with_values(ground_floor: 0, first_floor: 1, second_floor: 2, third_floor: 3, library: 4, staff_cafeteria: 5, auditorium: 6, parking_lot: 7) }
  end

  describe 'Associations' do
    it { should have_many(:sessions) }
  end
end
```

Figure 5.142: Venue Model Unit Tests

Question Model

The unit tests that have been completed for the Question model are as follows.

Attribute Validations



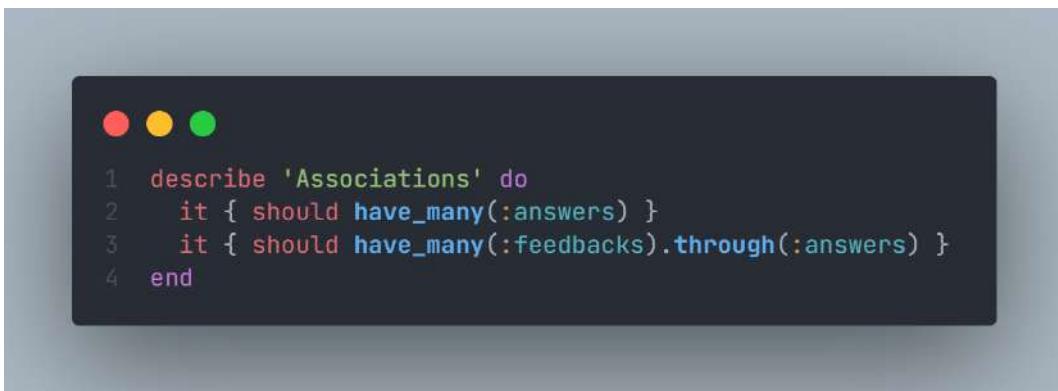
```
describe 'Validations' do
  it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
  it { should validate_presence_of(:title) }
end
```

Figure 5.143: Question Model - Validation Unit Tests

Presence validation: The validate_presence_of test is applied to make sure a Question instance cannot be saved without essential attributes. This is shown in our test suite by validate_presence_of(:title), ensuring each question must have a title.

Enumeration validation: The define_enum_for method is used for validating that the named values of an attribute correctly map to integer values in the database. In our suite, define_enum_for(:status).with_values(active: 0, inactive: 1) verifies the status attribute accurately maps 'active' and 'inactive' to 0 and 1, respectively.

Model Associations



```
1 describe 'Associations' do
2   it { should have_many(:answers) }
3   it { should have_many(:feedbacks).through(:answers) }
4 end
```

Figure 5.144: Question Model - Association Unit Tests

One-to-many relationships: The `have_many` test verifies that one instance of the Question model can be associated with multiple instances of another model. In our test suite, `it should have_many(:answers)` confirms that a Question can have multiple Answers.

Many-to-many relationships: The `have_many...through` test is employed when a Question instance can be associated with many instances of another model via a third model. In our test suite, `it should have_many(:feedbacks).through(:answers)` validates that a Question can be associated with many Feedbacks through Answers.



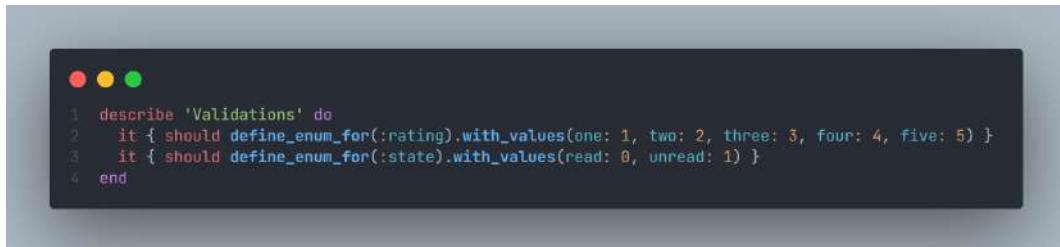
```
1 require 'rails_helper'
2
3 RSpec.describe Question, type: :model do
4   subject { build(:question) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
12    it { should validate_presence_of(:title) }
13  end
14
15  describe 'Associations' do
16    it { should have_many(:answers) }
17    it { should have_many(:feedbacks).through(:answers) }
18  end
19end
20
```

Figure 5.145: Question Model Unit Tests

Feedback Model

The unit tests that have been completed for the Feedback model are as follows.

Attribute Validations



```
● ● ●
1 describe 'Validations' do
2   it { should define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5) }
3   it { should define_enum_for(:state).with_values(read: 0, unread: 1) }
4 end
```

Figure 5.146: Feedback Model - Validation Unit Tests

Enumeration validation: The `define_enum_for` method is employed for validating the mapping of named values of an attribute to integer values in the database. In our suite, `define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5)` ensures the rating attribute correctly maps the terms 'one' through 'five' to 1 through 5, respectively. Similarly, `define_enum_for(:state).with_values(read: 0, unread: 1)` verifies the state attribute accurately maps 'read' and 'unread' to 0 and 1, respectively.

Model Associations



```
● ● ●
1 describe 'Associations' do
2   it { should belong_to(:user) }
3   it { should belong_to(:session) }
4   it { should have_many(:answers) }
5   it { should have_many(:questions).through(:answers) }
6 end
```

Figure 5.147: Feedback Model - Association Unit Tests

One-to-one relationships: The belong_to test validates that an instance of the Feedback model can be associated with one instance of another model. In our test suite, it should belong_to(:user) and it should belong_to(:session) confirm that a Feedback instance can be related to a specific User and Session.

One-to-many relationships: The have_many test confirms that one instance of the Feedback model can be associated with multiple instances of another model. In our suite, it should have_many(:answers) verifies that a Feedback can have multiple Answers.

Many-to-many relationships: The have_many...through test is used when a Feedback instance can be associated with many instances of another model via a third model. In our suite, it should have_many(:questions).through(:answers) validates that a Feedback can be associated with many Questions through Answers.



```
1 require 'rails_helper'
2
3 RSpec.describe Feedback, type: :model do
4   subject { build(:feedback) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5) }
12    it { should define_enum_for(:state).with_values(read: 0, unread: 1) }
13  end
14
15  describe 'Associations' do
16    it { should belong_to(:user) }
17    it { should belong_to(:session) }
18    it { should have_many(:answers) }
19    it { should have_many(:questions).through(:answers) }
20  end
21 end
22
```

Figure 5.148: Feedback Model Unit Tests

Material Model

The unit tests that have been completed for the Material model are as follows.

Attribute Validations



```
1 describe 'Validations' do
2   it { should validate_presence_of(:key) }
3   it { should validate_uniqueness_of(:key) }
4   it { should validate_presence_of(:bucket) }
5   it { should validate_presence_of(:content_type) }
6   it { should validate_presence_of(:name) }
7   it { should validate_presence_of(:url) }
8   it { should validate_presence_of(:size) }
9   it { should define_enum_for(:service).with_values(gcs: 0) }
10  it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
11  it { should define_enum_for(:visibility).with_values(visible: 0, invisible: 1) }
12 end
```

Figure 5.149: Material Model - Validation Unit Tests

Presence validation: The validate_presence_of tests ensure that certain attributes must contain values. For example, validate_presence_of(:name), validate_presence_of(:url), validate_presence_of(:size), etc., verifies that a Material instance cannot be saved without these respective attributes.

Uniqueness validation: These tests validate that the value of certain attributes is unique across all instances of a model, ensuring that no two materials can share the same value of such an attribute. For instance, validate_uniqueness_of(:key) asserts that each Material has a unique key.

Enumeration validation: We use define_enum_for to validate enumerations, where a set of named values correspond to integer values in the database. For example, define_enum_for(:service).with_values(gcs: 0), define_enum_for(:status).with_values(active: 0, inactive: 1), and define_enum_for(:visibility).with_values(visible: 0, invisible: 1) confirm that the service, status, and visibility attributes accurately map the respective terms to 0 and 1.

Model Associations



```
1 describe 'Associations' do
2   it { should belong_to(:user) }
3   it { should belong_to(:session) }
4 end
```

Figure 5.150: Material Model - Association Unit Tests

One-to-one relationships: The belong_to test validates that an instance of the Material model can be associated with one instance of another model. In our test suite, it should belong_to(:user) and it should belong_to(:session) confirm that a Material instance can be associated with a specific User and Session.



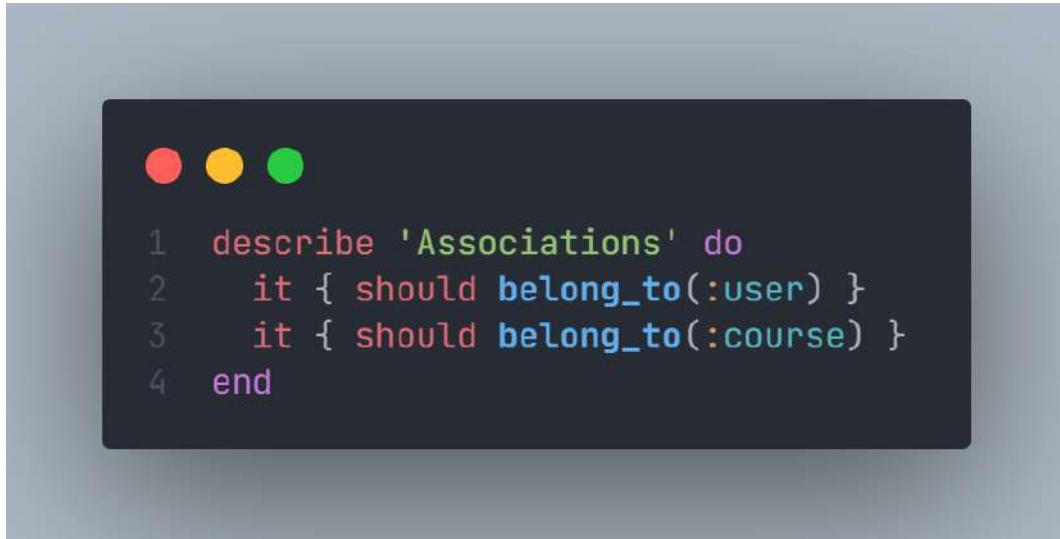
```
1 require 'rails_helper'
2
3 RSpec.describe Material, type: :model do
4   subject { build(:material) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should validate_presence_of(:key) }
12    it { should validate_uniqueness_of(:key) }
13    it { should validate_presence_of(:bucket) }
14    it { should validate_presence_of(:content_type) }
15    it { should validate_presence_of(:name) }
16    it { should validate_presence_of(:url) }
17    it { should validate_presence_of(:size) }
18    it { should define_enum_for(:service).with_values(gcs: 0) }
19    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
20    it { should define_enum_for(:visibility).with_values(visible: 0, invisible: 1) }
21  end
22
23  describe 'Associations' do
24    it { should belong_to(:user) }
25    it { should belong_to(:session) }
26  end
27end
28
```

Figure 5.151: Material Model Unit Tests

Enrollment Model

The unit tests that have been completed for the Session model are as follows.

Model Associations

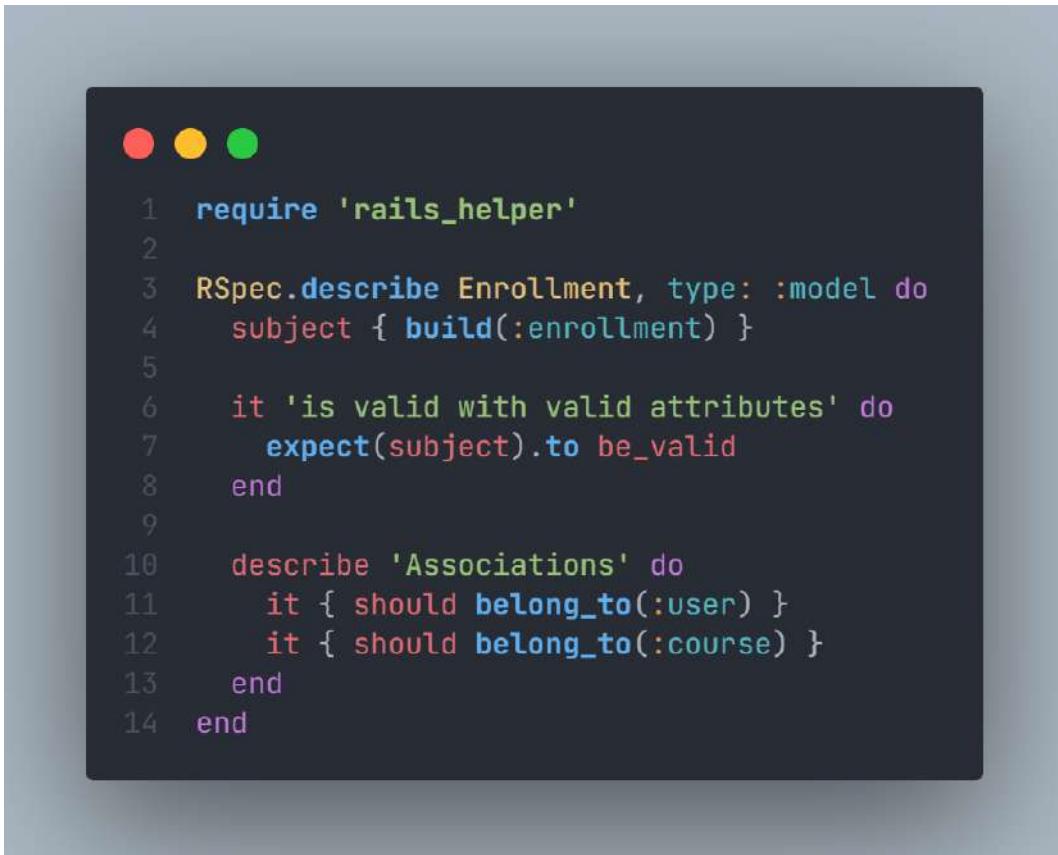


A screenshot of a terminal window with a dark background. At the top left are three colored circles: red, yellow, and green. Below them is a block of RSpec test code:

```
1 describe 'Associations' do
2   it { should belong_to(:user) }
3   it { should belong_to(:course) }
4 end
```

Figure 5.152: Enrollment Model - Association Unit Tests

One-to-one relationships: The belong_to test confirms that an instance of the Enrollment model can be associated with one instance of another model. In our test suite, it should belong_to(:user) and it should belong_to(:course) ascertain that an Enrollment instance can be linked with a particular User and Course.



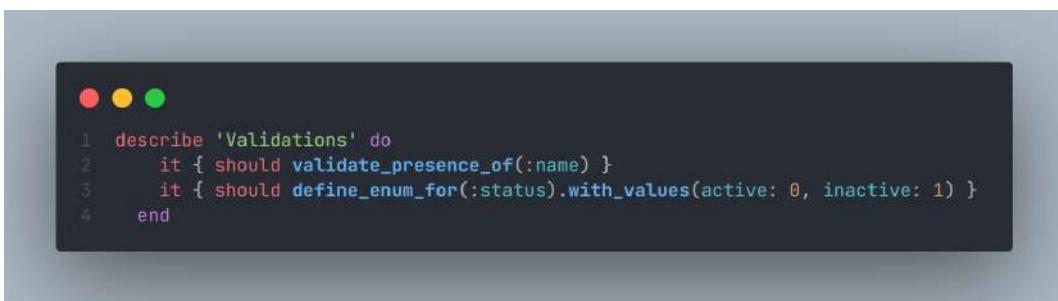
```
● ● ●
1 require 'rails_helper'
2
3 RSpec.describe Enrollment, type: :model do
4   subject { build(:enrollment) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Associations' do
11    it { should belong_to(:user) }
12    it { should belong_to(:course) }
13  end
14 end
```

Figure 5.153: Enrollment Model Unit Tests

Course Model

The unit tests that have been completed for the Course model are as follows.

Attribute Validations



```
● ● ●
1 describe 'Validations' do
2   it { should validate_presence_of(:name) }
3   it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
4 end
```

Figure 5.154: Course Model - Validation Unit Tests

Presence validation: The validate_presence_of test ensures that certain attributes are not empty. For example, validate_presence_of(:name) ensures that a Course instance cannot be saved without a name attribute.

Enumeration validation: The `define_enum_for` test validates enumerations, ensuring that a specific attribute maps certain names to integer values in the database. For example, `define_enum_for(:status).with_values(active: 0, inactive: 1)` validates that the `status` attribute maps 'active' to 0 and 'inactive' to 1.

Model Associations



```
1 describe 'Associations' do
2   it { should have_many(:enrollments) }
3   it { should have_many(:users).through(:enrollments) }
4 end
```

Figure 5.155: Course Model - Association Unit Tests

One-to-many relationships: The `have_many` test verifies that an instance of the Course model can be associated with multiple instances of another model. In our suite, `it should have_many(:enrollments)` confirms that a Course can have multiple Enrollments.

Many-to-many relationships: The `have_many...through` test validates relationships where a Course instance can be associated with many instances of another model through a third model. In our test suite, `it should have_many(:users).through(:enrollments)` asserts that a Course can be associated with many Users through Enrollments.



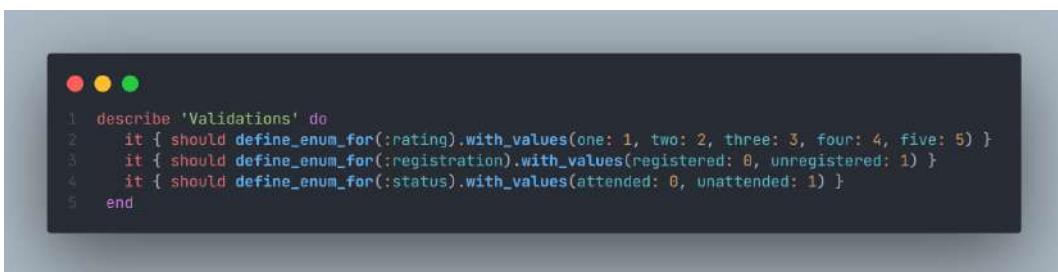
```
1 require 'rails_helper'
2
3 RSpec.describe Course, type: :model do
4   subject { build(:course) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should validate_presence_of(:name) }
12    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
13  end
14
15  describe 'Associations' do
16    it { should have_many(:enrollments) }
17    it { should have_many(:users).through(:enrollments) }
18  end
19 end
20
```

Figure 5.156: Course Model Unit Tests

Attendance Model

The unit tests that have been completed for the Attendance model are as follows.

Attribute Validations



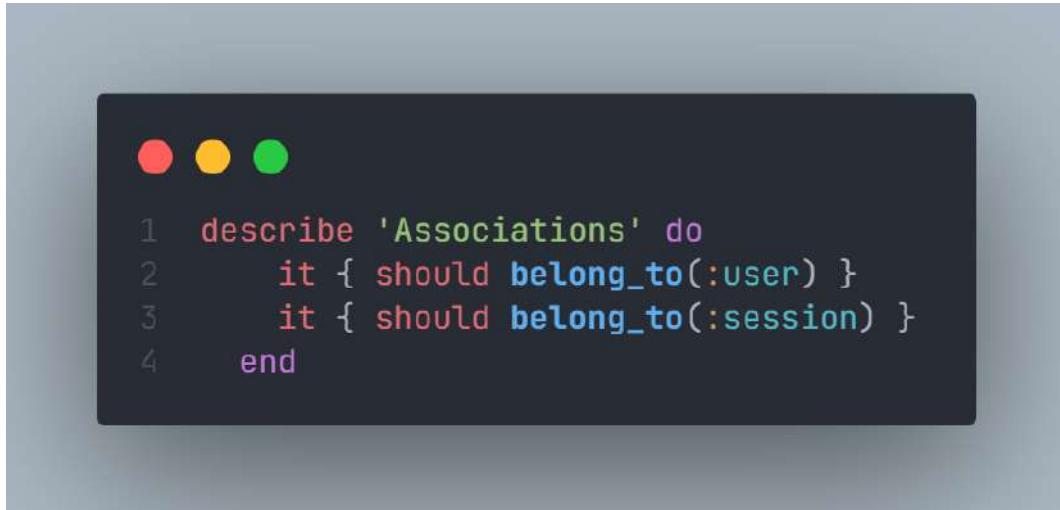
```
1 describe 'Validations' do
2   it { should define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5) }
3   it { should define_enum_for(:registration).with_values(registered: 0, unregistered: 1) }
4   it { should define_enum_for(:status).with_values(attended: 0, unattended: 1) }
5 end
```

Figure 5.157: Attendance Model - Validation Unit Tests

Enumeration validation: This model utilizes `define_enum_for` to validate three enumerations. The first test, `define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5)`, ensures that the rating attribute maps the numbers from one to five correctly. The second test, `define_enum_for(:registration).with_values(registered: 0, unregistered: 1)`, verifies that the registration attribute maps 'registered' to 0 and 'unregistered' to 1. The last test, `define_enum_for(:status).with_values(attended: 0,`

unattended: 1), confirms that the status attribute maps 'attended' to 0 and 'unattended' to 1.

Model Associations



```
● ● ●

1 describe 'Associations' do
2   it { should belong_to(:user) }
3   it { should belong_to(:session) }
4 end
```

Figure 5.158: Attendance Model - Association Unit Tests

One-to-one relationships: This model has two belong_to tests to confirm the association between the Attendance model and the User and Session models. The it should belong_to(:user) and it should belong_to(:session) tests ensure that each Attendance instance is associated with a single User and a single Session instance.



```
● ● ●

1 require 'rails_helper'
2
3 RSpec.describe Attendance, type: :model do
4   subject { build(:attendance) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should define_enum_for(:rating).with_values(one: 1, two: 2, three: 3, four: 4, five: 5) }
12    it { should define_enum_for(:registration).with_values(registered: 0, unregistered: 1) }
13    it { should define_enum_for(:status).with_values(attended: 0, unattended: 1) }
14  end
15
16  describe 'Associations' do
17    it { should belong_to(:user) }
18    it { should belong_to(:session) }
19  end
20 end
21
```

Figure 5.159: Attendance Model Unit Tests

Answer Model

The unit tests that have been completed for the Answer model are as follows.

Attribute Validations



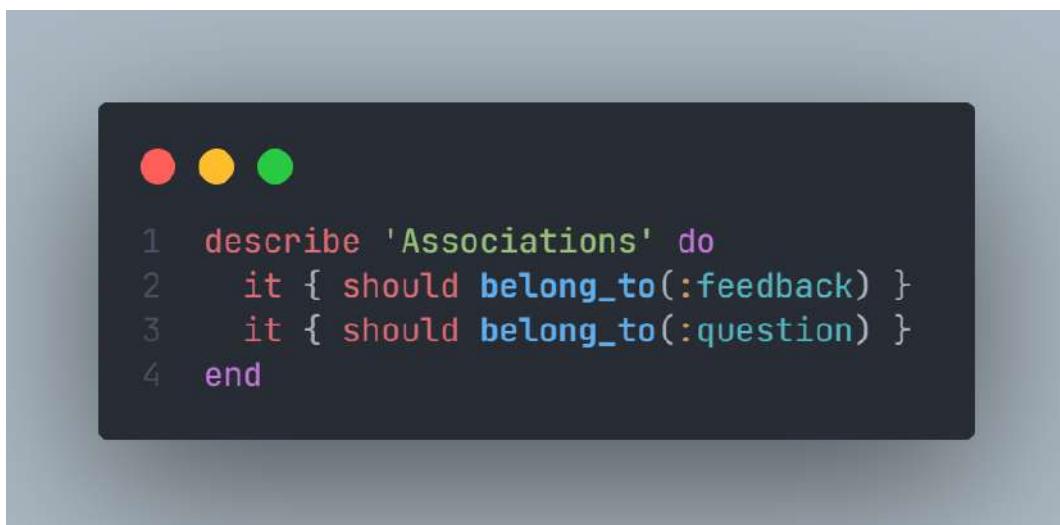
```
● ● ●
1 describe 'Validations' do
2   it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
3   it { should validate_presence_of(:content) }
4 end
```

Figure 5.160: Answer Model - Validation Unit Tests

Presence validation: The presence validation test validate_presence_of(:content) ensures that an Answer instance cannot be saved without the content attribute. This is crucial as each answer must have content to provide meaningful feedback.

Enumeration validation: This model uses define_enum_for to validate the status enumeration. The test define_enum_for(:status).with_values(active: 0, inactive: 1) confirms that the status attribute accurately maps the terms 'active' and 'inactive' to 0 and 1 respectively.

Model Associations



```
● ● ●
1 describe 'Associations' do
2   it { should belong_to(:feedback) }
3   it { should belong_to(:question) }
4 end
```

Figure 5.161: Answer Model - Association Unit Tests

One-to-one relationships: The Answer model has two belong_to tests that confirm its associations with the Feedback and Question models. The it should belong_to(:feedback) test ensures that each Answer instance is associated with a single Feedback instance, while the it should belong_to(:question) test verifies that each Answer is associated with a single Question instance.



```
● ● ●
1 require 'rails_helper'
2
3 RSpec.describe Answer, type: :model do
4   subject { build(:answer) }
5
6   it 'is valid with valid attributes' do
7     expect(subject).to be_valid
8   end
9
10  describe 'Validations' do
11    it { should define_enum_for(:status).with_values(active: 0, inactive: 1) }
12    it { should validate_presence_of(:content) }
13  end
14
15  describe 'Associations' do
16    it { should belong_to(:feedback) }
17    it { should belong_to(:question) }
18  end
19 end
20
```

Figure 5.162: Answer Model Unit Tests

Summary

Unit tests ensure that specific parts of the STDC application work as intended, which improves the application's overall reliability and stability. They assisted in the prevention of errors, the acceleration of development, and the preservation of application's quality.

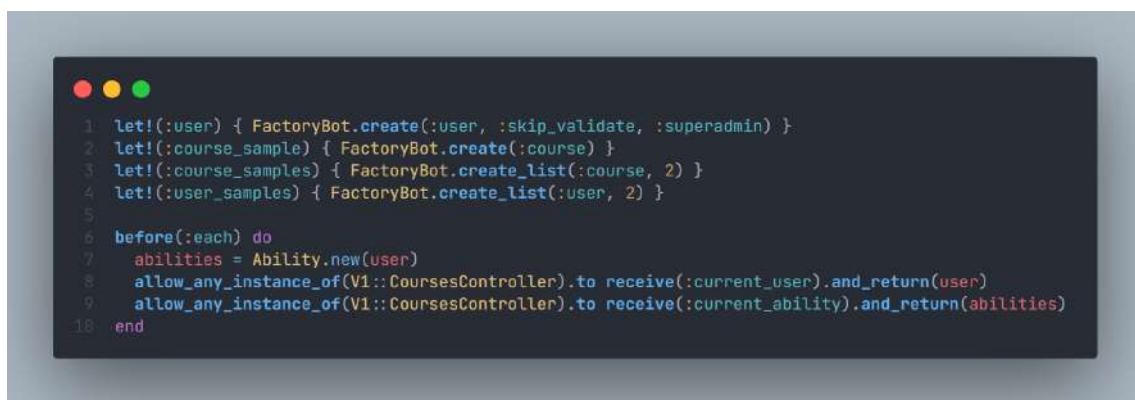
5.2.9 Integration Testing

Integration testing is a crucial aspect of testing of the Student Talent Development Center (STDC), offering a mechanism to combine individual units of the STDC application and tests them as a group. It is particularly valuable for catching issues that may have been missed during unit testing, and ensures that the various parts of the STDC application interact correctly. Since STDC's backend uses the Ruby on Rails framework, these components are represented by the controllers. Here, we will explore the integration testing strategies applied to one of the key controllers (Course). The testing is facilitated by the RSpec testing framework, enhanced by the RSwag gem, which aids in creating clean, readable, and effective test cases as well as swagger documentations based on the integration test cases.

Please note that only Course integration test has been highlighted below, however the rest of the integration tests can be found in the “stdc-api” repository that is available on github.

Common Testing Aspects (Setups)

Below here are line that you might see being repeated in almost all the test cases, for now, the course integration test is taken as an example, however the rest should be the same.



A screenshot of a terminal window showing RSpec setup code. The code is as follows:

```
1 let!(:user) { FactoryBot.create(:user, :skip_validate, :superadmin) }
2 let!(:course_sample) { FactoryBot.create(:course) }
3 let!(:course_samples) { FactoryBot.create_list(:course, 2) }
4 let!(:user_samples) { FactoryBot.create_list(:user, 2) }
5
6 before(:each) do
7   abilities = Ability.new(user)
8   allow_any_instance_of(V1::CoursesController).to receive(:current_user).and_return(user)
9   allow_any_instance_of(V1::CoursesController).to receive(:current_ability).and_return(abilities)
10 end
```

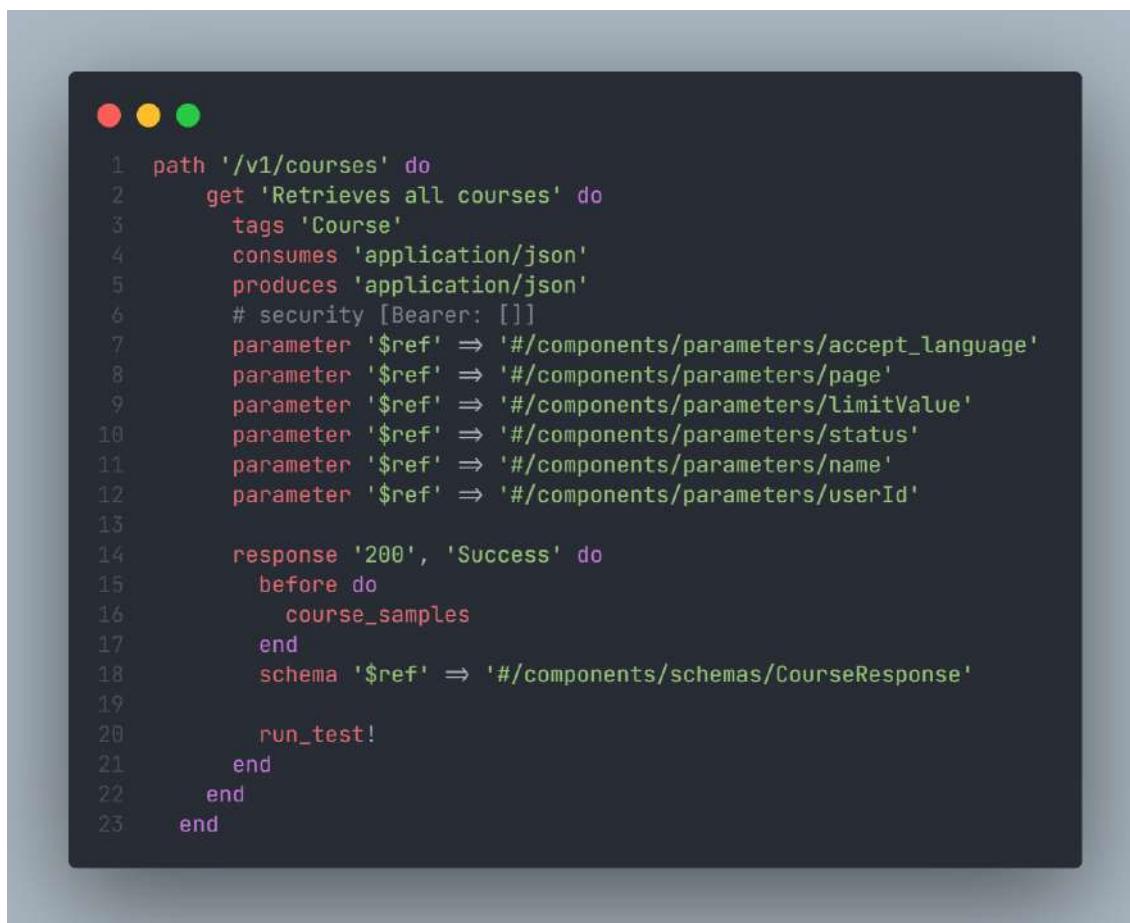
Figure 5.163: Automated Integration Testing - Setup

1. `let!(:user) FactoryBot.create(:user, :skip_validate, :superadmin)` : This line is using FactoryBot, a testing utility, to create a user. The symbols `:skip_validate` and `:superadmin` are traits defined in the FactoryBot configurations for user creation. This line creates a dummy user with the role of the superadmin to have access to everything. Additionally, validation for this user is skip on creation since the data used for creating the user are generated from Faker and do not reflect/represent real data.
2. `let!(:course_sample) FactoryBot.create(:course)` : This line is creating a single Course instance using FactoryBot which is used later on for testing endpoints that return one single object, such as show, create, update, etc.
3. `let!(:course_samples) FactoryBot.create_list(:course, 2)` : This line is using FactoryBot to create a list of two Course instances which are used later on for testing endpoints that return more than one object (a collection of objects) such as index endpoint.
4. The `before(:each)` block is run before each test case (each it block). In this particular application, almost all `before(:each)` block is creating a new instance of Ability with the user created earlier and then stubbing (or mocking) some methods:
5. `abilities = Ability.new(user)`: This line creates a new instance of Ability, a class to allow for user permissions and roles, using the previously created user with the help of the cancancan gem.
6. `allow_any_instance_of(V1::CoursesController).to receive(:current_user).and_return(user)`: This line is stubbing the `current_user` method on any instance of V1::CoursesController to return the user created earlier. This allows you to control the current user in the context of these tests, without having to go through the normal process of signing in or validation since this is a testing process.

- allow_any_instance_of(V1::CoursesController).to receive(:current_ability).and_return(abilities): Similar to the previous line, this line is stubbing the current_ability method on any instance of V1::CoursesController to return the abilities object created earlier to grant the user permission.

This setup ensures that each test runs in the context of a specific user (a superadmin, in this case) with a specific set of abilities, and with some specific course and user instances available. Please note that we only took Course as an example, other classes also have this setup for their integration tests.

Course Model



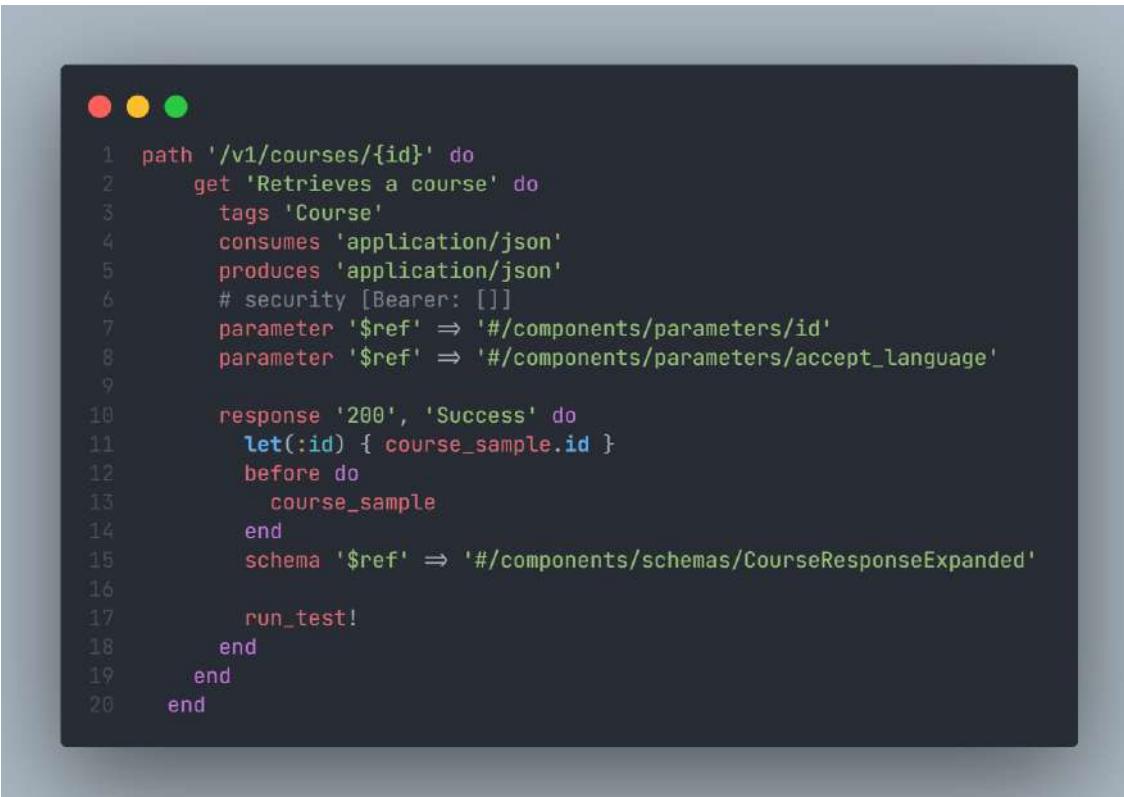
```

1 path '/v1/courses' do
2   get 'Retrieves all courses' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/accept_language'
8     parameter '$ref' => '#/components/parameters/page'
9     parameter '$ref' => '#/components/parameters/limitValue'
10    parameter '$ref' => '#/components/parameters/status'
11    parameter '$ref' => '#/components/parameters/name'
12    parameter '$ref' => '#/components/parameters/userId'
13
14    response '200', 'Success' do
15      before do
16        course_samples
17      end
18      schema '$ref' => '#/components/schemas/CourseResponse'
19
20      run_test!
21    end
22  end
23end

```

Figure 5.164: Course Integration Testing - Get All Courses

GET /v1/courses: This test case is checking if the application correctly retrieves all the courses. If the courses are successfully retrieved, a status of 200 and a JSON object containing all courses should be returned.



```
1 path '/v1/courses/{id}' do
2   get 'Retrieves a course' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/id'
8     parameter '$ref' => '#/components/parameters/accept_language'
9
10    response '200', 'Success' do
11      let(:id) { course_sample.id }
12      before do
13        course_sample
14      end
15      schema '$ref' => '#/components/schemas/CourseResponseExpanded'
16
17      run_test!
18    end
19  end
20end
```

Figure 5.165: Course Integration Testing - Get Course By ID

GET /v1/courses/{id}: This test case verifies that a course can be retrieved individually using its id. If the specified course exists, it should return a 200 status and the JSON object of the course.



```
1 path '/v1/courses' do
2   post 'Creates a course' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/accept_language'
8
9     parameter name: :data, in: :body, schema: { '$ref' => '#/components/schemas/CourseRequest' }
10
11    response '201', 'Success' do
12      let(:data) { { data: FactoryBot.attributes_for(:course) } }
13      schema '$ref' => '#/components/schemas/CourseResponseExpanded'
14
15      run_test!
16    end
17  end
18end
```

Figure 5.166: Course Integration Testing - Create Courses

POST /v1/courses: This test case is checking if a course can be successfully created. The test data for the course is passed in the request body. If the course is created successfully, it should return a 201 status and the JSON object of the created course.



```
1 path '/v1/courses/{id}' do
2   put 'Updates a course' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/id'
8     parameter '$ref' => '#/components/parameters/accept_language'
9
10    parameter name: :data, in: :body, schema: { '$ref' => '#/components/schemas/CourseRequest' }
11
12    response '200', 'Success' do
13      let(:id) { course_sample.id }
14      let(:data) { { data: FactoryBot.attributes_for(:course) } }
15      schema '$ref' => '#/components/schemas/CourseResponseExpanded'
16
17      run_test!
18    end
19  end
20end
```

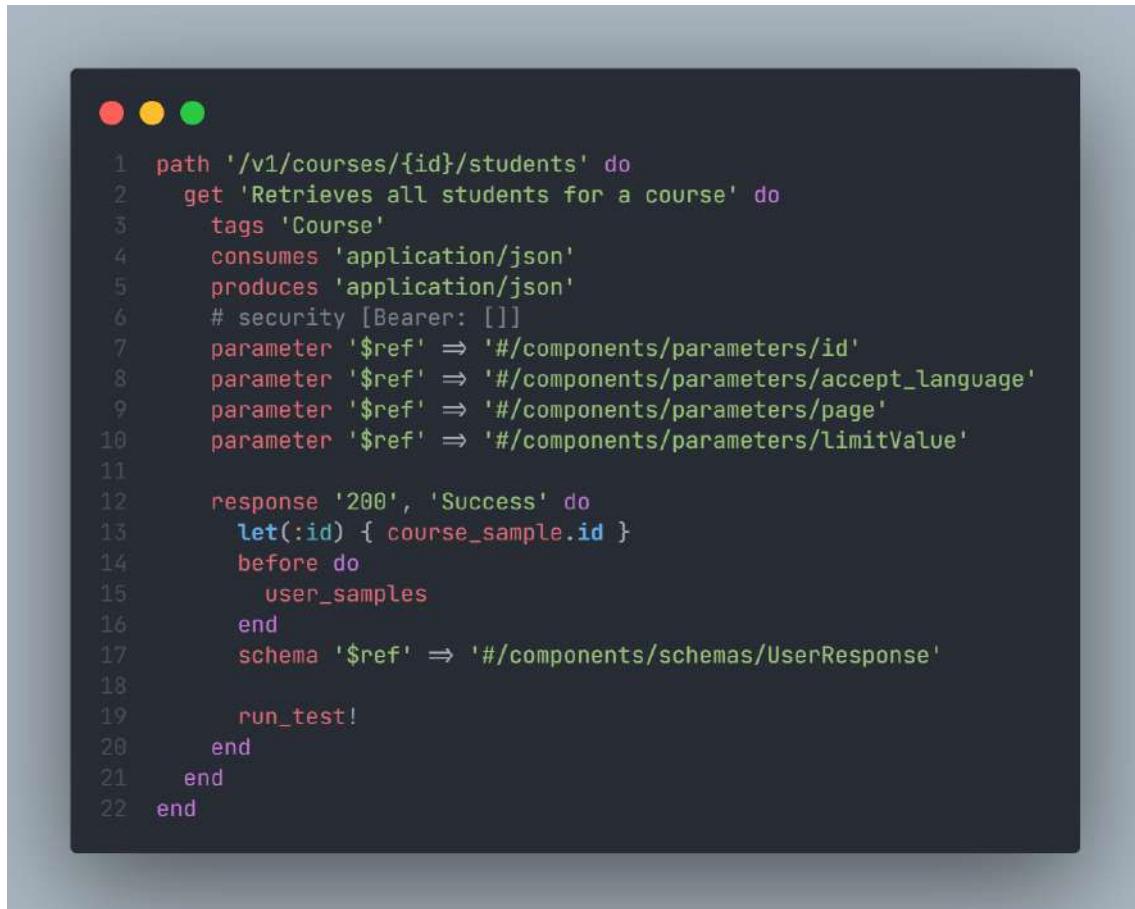
Figure 5.167: Course Integration Testing - Update Course

PUT /v1/courses/:id: This test case verifies that an existing course can be updated. The new data for the course is passed in the request body. If the course is updated successfully, it should return a 200 status and the JSON object of the updated course.

```
1 path '/v1/courses/all' do
2   get 'Retrieves all courses (unpaginated)' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/accept_language'
8     parameter '$ref' => '#/components/parameters/status'
9     parameter '$ref' => '#/components/parameters/name'
10    parameter '$ref' => '#/components/parameters/userId'
11
12    response '200', 'Success' do
13      before do
14        course_samples
15      end
16      schema '$ref' => '#/components/schemas/CourseResponse'
17
18      run_test!
19    end
20  end
21 end
```

Figure 5.168: Course Integration Testing - Get All Courses With No Pagination

GET /v1/courses/all: This test case is checking if all the courses can be retrieved without pagination. If the courses are successfully retrieved, a status of 200 and a JSON object containing all courses should be returned.



```
1 path '/v1/courses/{id}/students' do
2   get 'Retrieves all students for a course' do
3     tags 'Course'
4     consumes 'application/json'
5     produces 'application/json'
6     # security [Bearer: []]
7     parameter '$ref' => '#/components/parameters/id'
8     parameter '$ref' => '#/components/parameters/accept_language'
9     parameter '$ref' => '#/components/parameters/page'
10    parameter '$ref' => '#/components/parameters/limitValue'
11
12    response '200', 'Success' do
13      let(:id) { course_sample.id }
14      before do
15        user_samples
16      end
17      schema '$ref' => '#/components/schemas/UserResponse'
18
19      run_test!
20    end
21  end
22end
```

Figure 5.169: Course Integration Testing - Get All Students

GET /v1/courses/id/students: This test case is to verify that all students for a particular course can be retrieved. If the students are successfully retrieved, a 200 status and a JSON object containing all students in the course should be returned.

```
1 path '/v1/courses/assign-tutor' do
2   before(:each) do
3     allow_any_instance_of(Enrollment).to receive(:save).and_return(true)
4   end
5
6   post 'Assigns a tutor to a course' do
7     tags 'Course'
8     consumes 'application/json'
9     produces 'application/json'
10    # security [Bearer: []]
11    parameter '$ref' => '#/components/parameters/accept_language'
12
13    parameter name: :data, in: :body, schema: {
14      type: :object,
15      properties: {
16        data: {
17          type: :object,
18          properties: {
19            courseId: { type: :integer },
20            userId: { type: :integer }
21          }
22        },
23        required: %w[courseId userId]
24      }
25    }
26
27    response '200', 'Success' do
28      let(:data) do
29        { data: {
30          courseId: course_sample.id,
31          userId: user.id
32        } }
33      end
34      schema '$ref' => '#/components/schemas/CourseResponseExpanded'
35
36      run_test!
37    end
38  end
39end
```

Figure 5.170: Course Integration Testing - Assign Tutor

POST /v1/courses/assign-tutor: This test case checks if a tutor can be assigned to a course. The courseId and userId are passed in the request body. If the tutor is successfully assigned, it should return a 200 status and the JSON object of the course with the assigned tutor.

Summary

Integration tests ensure that specific parts of the STDC application work as intended, which improves the application's overall reliability and stability. They assisted in the prevention of errors, the acceleration of development, and the preservation of application's quality.

5.2.10 Build Verification Testing

The purpose of build verification testing is to ensure that the STDC Web application build is functional and complies with all of the requirements. It is standard practice just before a new software version is released, build verification tests are done. The STDC's backend repository on GitHub is where the YAML file describing the CI/CD (continuous integration / continuous deployment) pipeline for the STDC can be found, and it is from there that the build verification testing is executed. Note that this test is an automated test methodology.

When a change is made to the main branch, also known as the production branch, the CI/CD pipeline is triggered. There are two jobs in the pipeline: testing and deploying. On an Ubuntu VM, the following operations constitute the Test job:

1. Pulls the latest code from the main branch.
2. Sets up the required Ruby and Node.js environments.
3. Caches the Yarn package manager as well as Ruby gems to speed up future builds.
4. Installs the project's dependencies using Yarn and Bundler.
5. Runs the project's tests using the Rails test framework.

All tests must pass before the pipeline moves on to the Deploy job, which deploys the application to production from the main branch.

There are several ways in which the application development process is improved by the build verification testing procedure:

- Early discovery of problems: The practice of build verification testing assists in the early detection of problems early on in the development cycle, hence preventing those problems from being introduced into the production environment. This helps to reduce the amount of time and cost required to fix issues that could arise later.
- A shorter time required to complete the feedback loop. The CI/CD pipeline

shortens the time required for developers to receive feedback on the quality of their code by automatically executing tests after each commit. This contributes to the enhancement of the code's quality as well as the reduction of the risk of introducing errors.

- Consistent benchmark for testing the code. This assists in increasing the quality of the code as a whole as well as eliminating conflicts amongst different version of the code base from different branches.
- The CI/CD pipeline automates the deployment process, which minimizes the risk of human mistake and makes the deployment process more efficient and dependable. One benefit of this automation is that the deployment process is enhanced.

In conclusion, the process of build verification testing is a vital component of the development cycle that contributes to the enhancement of the software's quality as well as the reduction of the amount of time required to resolve issues. The Continuous Integration and Continuous Deployment (CI/CD) pipeline that is supplied in the YAML file (Below here) automates the testing and deployment process. This provides the STDC development with faster feedback and a deployment procedure that is more simplified.

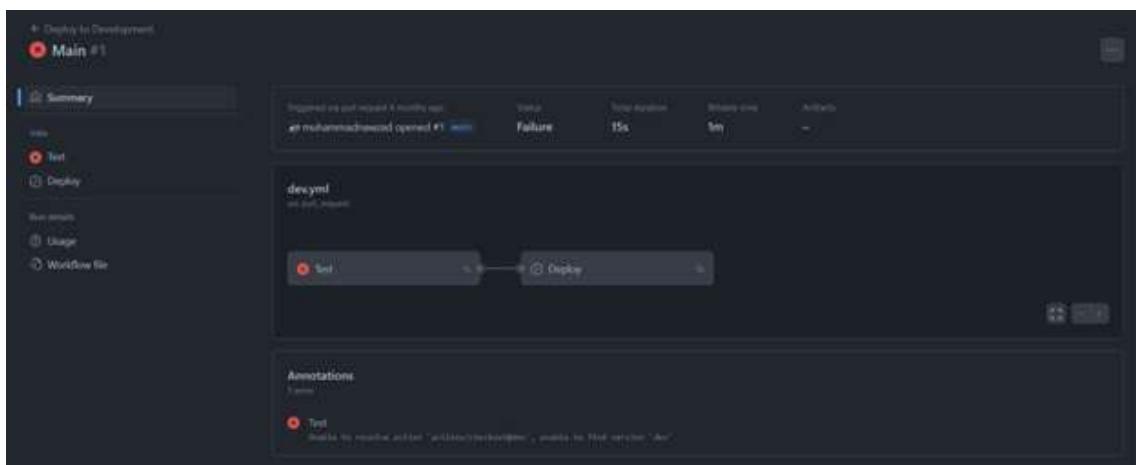


Figure 5.171: Build Verification Testing - Failed Scenario

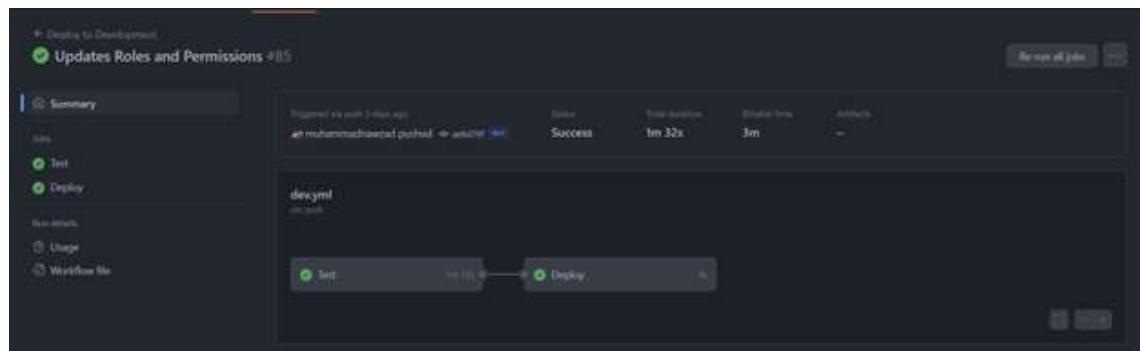


Figure 5.172: Build Verification Testing - Succeeded Scenario

```
1 name: Deploy to Production
2
3 on:
4   push:
5     branches:
6       - main
7
8 jobs:
9   Test:
10    runs-on: ubuntu-latest
11    services:
12      db:
13        image: postgres:15.1
14        env:
15          POSTGRES_USERNAME: postgres
16          POSTGRES_PASSWORD: mysecretpassword
17        ports:
18          - 5432:5432
19      redis:
20        image: bitnami/redis:7.0
21        ports:
22          - 6379:6379
23
24 steps:
25   - uses: actions/checkout@main
26
27   - name: Setup Ruby 3.2.0
28     uses: ruby/setup-ruby@v1
29     with:
30       ruby-version: 3.2.0
31
32   - name: Setup Node 19.0.1
33     uses: actions/setup-node@v3
34     with:
35       node-version: 19.0.1
36
37   - name: Get yarn cache
38     id: yarn-cache
39     run: echo "dir=$(yarn cache dir)" >>$GITHUB_OUTPUT
40
41   - name: Cache yarn
42     uses: actions/cache@v3
43     with:
44       path: ${{ steps.yarn-cache.outputs.dir }}
45       key: ${{ runner.os }}-yarn-${{ hashFiles('**/yarn.lock') }}
46       restore-keys: |
47         ${{ runner.os }}-yarn-
48
49   - name: Cache gems
50     uses: actions/cache@v3
51     with:
52       path: vendor/bundle
53       key: ${{ runner.os }}-gem-${{ hashFiles('**/Gemfile.lock') }}
54       restore-keys: |
55         ${{ runner.os }}-gem-
56
57   - name: Install dependencies
58     run: |
59       gem install bundler
60       bundle config path vendor/bundle
61       bundle install --jobs 4 --retry 3
62       yarn install --frozen-lockfile
63
64   - name: Run tests
65     env:
66       DATABASE_URL: "postgres://postgres:mysecretpassword@localhost:5432/stdc_api_test"
67       REDIS_URL: redis://localhost:6379/1
68       RAILS_ENV: test
69       RAILS_MASTER_KEY: ${{ secrets.RAILS_MASTER_KEY }}
70       PG_USER: postgres
71     run: |
72       bin/rails db:test:prepare
73       bin/rails test
74
75 Deploy:
76   if: ${{ github.event_name == 'push' && github.ref == 'refs/heads/main' }}
77   needs: Test
78   runs-on: ubuntu-latest
79
80   steps:
81     - uses: actions/checkout@main
```

Figure 5.173: Build Verification Testing - CI/CD YAML

5.2.11 Usability Testing

Testing for usability is an essential testing methodology that serves the purpose of ensuring that a product, application, or service is user-friendly, productive, and successful for the people who are intended to use it. Testing the usability of an application entails observing and collecting feedback from users as they engage with the product or application in order to discover areas that could use improvement. The Student Talent Development Center (STDC) Web Application will undergo a usability test in order to evaluate the user experience and identify any issues that may prevent users from achieving their goals or lead to frustration and confusion. This will be accomplished by understanding how users interact with the application.

Tests of usability can be performed in a variety of ways, including in-person or remote testing, task-based or exploratory testing, expert or user testing, and more. The usability test is carried out in two stages within the framework of the STDC. Initially, participants were given the application to try out for themselves, and then, after completing the test, they were given a questionnaire to complete in order to assess the program's level of usability.

The findings of a usability test revealed some interesting and useful information on the STDC. It contributed to the identification of areas in which the designs and the user experience may be improved. Testing for usability is an iterative process; however, due to time constraints, we were only able to complete one round of testing for usability. Despite this, the findings of the test could inform us on the overall design of the application, assisting us in identifying areas that could make STDC a more user-friendly and effective application.

In general, usability testing is a vital component of the testing process, as it contributes to the development of applications that successfully satisfy the requirements of their users and deliver a satisfying experience to those users.

Methodology

For this usability test, a survey-based approach was used to gather feedback from

participants. The survey was created using Microsoft Forms and included questions that focused on the ease of use, navigation, and overall user experience of the STDC application. The survey was distributed to a convenience sample of at least 08 participants who were selected based on their familiarity with the subject matter and potential use of the application.

Participants were asked to complete the survey based on their experience using the STDC application, which they accessed via a provided link. Participants were given approximately 15 minutes to complete the survey and an additional 30 to 60 mins to completely test the application.

The survey consisted of closed and open-ended questions. Closed-ended questions used a scale to measure the level of agreement or disagreement with statements related to the usability of the application. Open-ended questions allowed participants to provide more detailed feedback about specific aspects of the application that they found challenging or easy to use.

After the survey data was collected, the results were analyzed to identify common themes and issues that emerged across participants. Both quantitative and qualitative methods were used to analyze the data.

While we recognize that survey-based approaches may not capture all aspects of the user experience, we felt that it was a practical approach given our time constraints and the need to gather feedback from a diverse group of participants.

Participants

The survey was distributed to a convenience sample of at least 08 participants who were selected based on their familiarity with the subject matter and potential use of the application. Additionally, participants with higher tech-savviness were more favorable to be chose as participants to test the application.

Also due to the small number of staff at the STDC, only one participant who is a staff was able to test the application.

Tasks

The usability test is carried out in two stages within the framework of the STDC. Initially, participants were given the application to try out for themselves, and then, after completing the test, they were given a questionnaire to complete in order to assess the program's level of usability.

Questionnaires

The followings are the questions of the survey/questionnaire that were asked from the participants.

← Back Computer Mobile

May, 2023

Student Talent Development Center (STDC) Web Application

The following survey is being conducted to gather feedback on the usability of the STDC web application as a part of undergraduate final year project, with the goal of improving the user experience. Please take at max 12 mins to answer the following questions regarding your experience.

Start now

This content is created by the owner of the form. The data you submit will be sent to the form owner. Microsoft is not responsible for the privacy or security practices of its customers, including those of this form owner. Never give out your password.

Powered by Microsoft Forms | [Privacy and cookies](#) | [Terms of use](#)

Figure 5.174: Usability Testing Questionnaires - Guide

The screenshot shows a Microsoft Forms survey titled "Student Talent Development Center (STDC) Web Application". At the top, there are navigation links for "Back", "Computer" (which is selected), and "Mobile". Below the title, a note says "* Required".

Testing the Application

Please visit the following link to test the Student Talent Development Center's Web Application. Consider that the aim of this is to test the application's usability. Please navigate through the application thoroughly.

1. Have you tested the application? *

Yes
 No

Next

This content is created by the owner of the form. The data you submit will be sent to the form owner. Microsoft is not responsible for the privacy or security practices of its customers, including those of this form owner. Never give out your password.

Powered by Microsoft Forms | [Privacy and cookies](#) | [Terms of use](#)

Figure 5.175: Usability Testing Questionnaires - Section I

The screenshot shows the continuation of the Microsoft Forms survey titled "Student Talent Development Center (STDC) Web Application". At the top, there are navigation links for "Back", "Computer" (selected), and "Mobile". Below the title, a note says "* Required".

Usability Test Questionnaire

Upon finishing the above, please answer the following questions.

2. What is your current occupation? *

Student at UKH
 Staff/Admin at UKH
 Other

3. How would you rate your level of confidence in using your mobile phone / personal computer for browsing the internet?

4. Have you previously heard about the Student Talent Development Center? If yes, how have you interacted with STDC?

Yes, I am/was an admin there.
 Yes, I am/was a tutor there.

Figure 5.176: Usability Testing Questionnaires - Section II (1)

← Back Computer Mobile

4. Have you previously heard about the Student Talent Development Center? If yes, how + *

Have you interacted with STDC?

- Yes, I am/was an admin there.
- Yes, I am/was a tutor there.
- Yes, I am/was a student there.
- Yes, I was aware of its existence but have not engaged with the center.
- No, I have not heard of STDC prior to this questionnaire.

5. What device did you use for testing STDC web application? ← *

- Mobile Phone
- Laptop/PC

6. How easy or difficult was it to sign up for the application? + *

- Extremely difficult
- Somewhat difficult
- Neutral

Figure 5.177: Usability Testing Questionnaires - Section II (2)

← Back Computer Mobile

6. How easy or difficult was it to sign up for the application? + *

- Extremely difficult
- Somewhat difficult
- Neutral
- Somewhat not difficult
- Extremely not difficult

7. How easy or difficult was it to navigate through the application? + *

(How easy was it to find what you were looking for?)

- Extremely easy
- Somewhat easy
- Neutral
- Somewhat not easy
- Extremely not easy

Figure 5.178: Usability Testing Questionnaires - Section II (3)

← Back Computer Mobile

8. What are your thoughts on the design and layout? *

- Very appealing and easy to use
- Somewhat appealing, but needs improvement in usability
- No opinion on the design and layout
- Needs significant improvement in both design and usability
- Not appealing at all and difficult to use

9. On the scale of (1..10), how did you find the information provided by the application? (1 being not informative at all and 10 being extremely informative)

1 2 3 4 5 6 7 8 9 10

10. How easy was it to find the available courses and sessions? *

- Extremely easy
- Somewhat easy
- Neutral

Figure 5.179: Usability Testing Questionnaires - Section II (4)

← Back Computer Mobile

10. How easy was it to find the available courses and sessions? *

- Extremely easy
- Somewhat easy
- Neutral
- Somewhat not easy
- Extremely not easy

11. How difficult was it to find time and venue for a session? *

- Extremely difficult
- Somewhat difficult
- Neutral
- Somewhat not difficult
- Extremely not difficult

12. How did you feel about providing feedback about sessions? *

Figure 5.180: Usability Testing Questionnaires - Section II (5)

← Back Computer Mobile

12. How did you feel about providing feedback about sessions? *

Very confident
 Confident
 Not so confident
 Uneasy

13. How would you describe your overall experience? *

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

14. What did you like the most about using this product?

Enter your answer

15. How can you increase the product's ease of use?

Enter your answer

Figure 5.181: Usability Testing Questionnaires - Section II (6)

← Back Computer Mobile

15. How can you increase the product's ease of use?

Enter your answer

16. What did you like the least about using this product?

Enter your answer

17. What, if anything, caused you frustration?

Enter your answer

[Back](#) [Submit](#)

This content is created by the owner of the form. The data you submit will be sent to the form owner. Microsoft is not responsible for the privacy or security practices of its customers, including those of this form owner. Never give out your password.

Powered by Microsoft Forms | [Privacy and cookies](#) | [Terms of use](#)

Figure 5.182: Usability Testing Questionnaires - Section II (7)

Results and Analysis

Below here is a brief analysis of the result of each question. It is worth noting that not all the details may have been highlighted.

Question 1: Have you tested the application?

1. Have you tested the application?

[More Details](#)

● Yes	9
● No	0



Figure 5.183: Usability Testing Results - Question 1

100% of participants tested the application, this indicates that all responses given are based on actual usage experience and also it provides valuable feedback on the usability of the application.

Question 2: What is your current occupation?

2. What is your current occupation?

[More Details](#)

● Student at UKH	8
● Staff/Admin at UKH	1
● Other	0



Figure 5.184: Usability Testing Results - Question 2

Most of the participants are students, with only one participant identifying as a Staff / Admin. This gives a good perspective from the main user base (students), but the limited feedback from staff/admin may not fully represent their experience.

Question 3: How would you rate your level of confidence in using your mobile phone / personal computer for browsing the internet?

3. **How would you rate your level of confidence in using your mobile phone / personal computer for browsing the internet?**

[More Details](#)

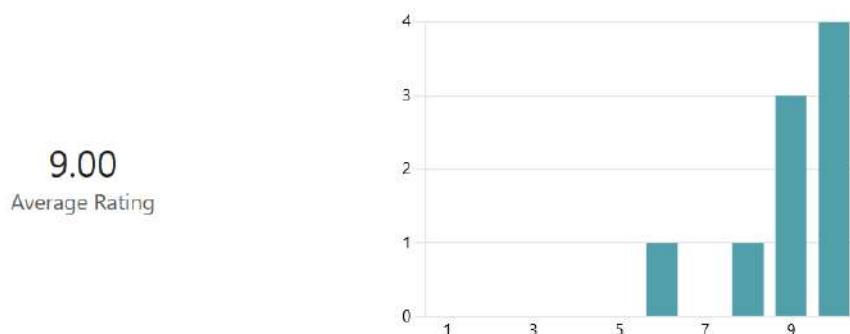


Figure 5.185: Usability Testing Results - Question 3

The majority of participants rated their level of confidence as very high (9 or 10). This indicates that the user base is tech-savvy and comfortable using digital platforms. The average rating of 9.0 suggests that most issues identified during the testing are likely related to the application itself rather than a lack of technical skills from the users.

Question 4: Have you previously heard about the Student Talent Development Center? If yes, how have you interacted with STDC?

4. **Have you previously heard about the Student Talent Development Center? If yes, how have you interacted with STDC?**

[More Details](#)

- Yes, I am/was an admin there. 0
- Yes, I am/was a tutor there. 2
- Yes, I am/was a student there. 2
- Yes, I was aware of its existence ... 4
- No, I have not heard of STDC pri... 1

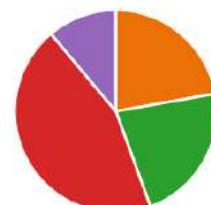


Figure 5.186: Usability Testing Results - Question 4

Most participants were aware of the STDC's existence but had not engaged with it. This shows that while the STDC has visibility, it might need to work on its engagement strategies to better reach and interact with its potential students which in a way indicates the importance of this system.

Question 5: What device did you use for testing STDC web application?

5. What device did you use for testing STDC web application?

[More Details](#)

●	Mobile Phone	2
●	Laptop/PC	7

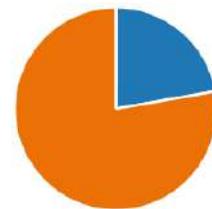


Figure 5.187: Usability Testing Results - Question 5

Most participants used a Laptop/PC to test the application. This information could be used to prioritize design and functionality aspects tailored towards desktop users in the future.

Question 6: How easy or difficult was it to sign up for the application?

6. How easy or difficult was it to sign up for the application?

[More Details](#)

●	Extremely difficult	0
●	Somewhat difficult	1
●	Neutral	0
●	Somewhat not difficult	3
●	Extremely not difficult	5

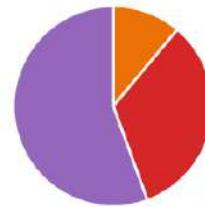


Figure 5.188: Usability Testing Results - Question 6

Most participants found it not difficult to sign up, suggesting the sign up process is intuitive and user friendly, however there are concerns still regarding the sign up which we will talk about in details in the next section (Recommendations).

Question 7: How easy or difficult was it to navigate through the application?

7. How easy or difficult was it to navigate through the application?

[More Details](#)

- | | |
|----------------------|---|
| ● Extremely easy | 3 |
| ● Somewhat easy | 5 |
| ● Neutral | 1 |
| ● Somewhat not easy | 0 |
| ● Extremely not easy | 0 |

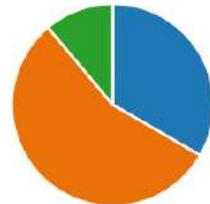


Figure 5.189: Usability Testing Results - Question 7

Overall, participants found the application relatively easy to navigate. This indicates the application's layout and navigation structure are well designed.

Question 8: What are your thoughts on the design and layout?

8. What are your thoughts on the design and layout?

[More Details](#)

- | | |
|--|---|
| ● Very appealing and easy to use | 7 |
| ● Somewhat appealing, but needs... | 2 |
| ● No opinion on the design and la... | 0 |
| ● Needs significant improvement ... | 0 |
| ● Not appealing at all and difficul... | 0 |

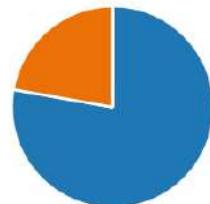


Figure 5.190: Usability Testing Results - Question 8

Most participants found the design appealing and easy to use. However, a minority suggested that it could benefit from usability improvements. This might indicate some minor issues that could be smoothed out to further enhance the user experience for which more details need to be collected from the users to see what are the specific issues.

Question 9: On the scale of (1..10), how did you find the information provided by the application?

9. **On the scale of (1..10), how did you find the information provided by the application?**

[More Details](#)

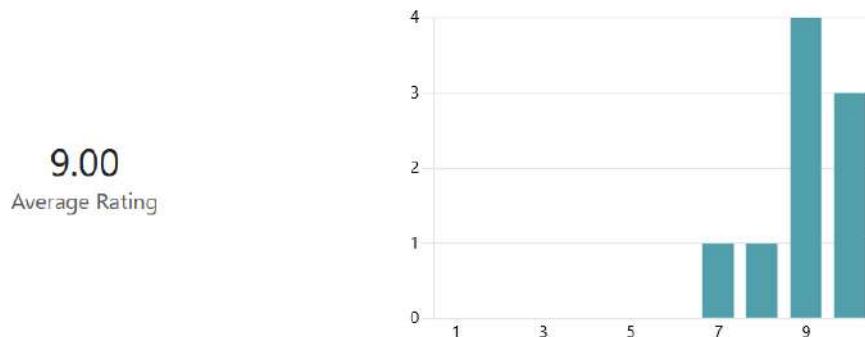


Figure 5.191: Usability Testing Results - Question 9

The high average rating indicates that participants found the information provided by the application to be useful and adequate.

Question 10: How easy was it to find the available courses and sessions?

10. **How easy was it to find the available courses and sessions?**

[More Details](#)

Extremely easy	6
Somewhat easy	2
Neutral	1
Somewhat not easy	0
Extremely not easy	0



Figure 5.192: Usability Testing Results - Question 10

Most participants found it extremely easy to find the available courses and sessions, indicating that these key features are accessible and well-implemented.

Question 11: How difficult was it to find time and venue for a session?

11. How difficult was it to find time and venue for a session?

[More Details](#)

● Extremely difficult	1
● Somewhat difficult	0
● Neutral	0
● Somewhat not difficult	3
● Extremely not difficult	5

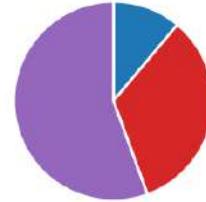


Figure 5.193: Usability Testing Results - Question 11

Most participants did not find it difficult to find the time and venue for a session, implying that this aspect of the application is well designed.

Question 12: How did you feel about providing feedback about sessions?

12. How did you feel about providing feedback about sessions?

[More Details](#)

● Very confident	5
● Confident	4
● Not so confident	0
● Uneasy	0

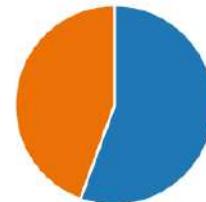


Figure 5.194: Usability Testing Results - Question 12

All participants felt confident or very confident in providing feedback about sessions. This shows that the feedback system is intuitive and user-friendly.

Question 13: How would you describe your overall experience?

13. **How would you describe your overall experience?**

[More Details](#)

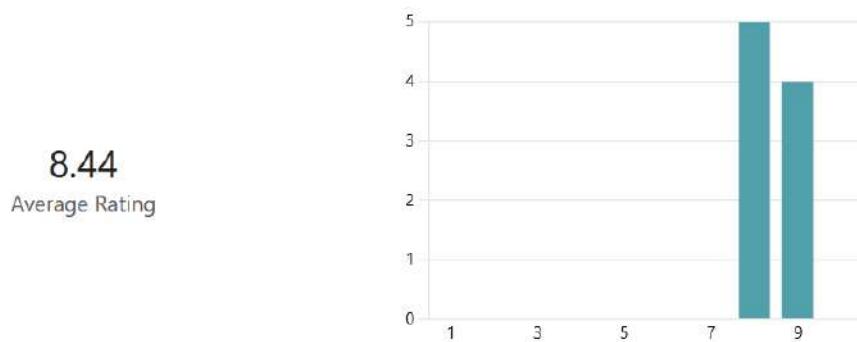


Figure 5.195: Usability Testing Results - Question 13

With an average rating of 8.44, participants had a positive overall experience with the application.

Question 14: What did you like the most about using this product?

X

14. What did you like the most about using this product?

6 Responses

ID ↑	Name	Responses
1	anonymous	It is very easy to use
2	anonymous	User-friendly, accessibility, provides everything a tutor center needs
3	anonymous	The design.
4	anonymous	A
5	anonymous	Very smooth
6	anonymous	The sign up process.

Figure 5.196: Usability Testing Results - Question 14

Participants particularly highlighted the ease of use the design that they liked the most..

Question 15: How can you increase the product's ease of use?

ID ↑	Name	Responses
1	anonymous	More back and forward access buttons
2	anonymous	Forward and backward buttons
3	anonymous	A

Figure 5.197: Usability Testing Results - Question 15

A few participants suggested the inclusion of more back and forward access buttons to improve navigation which seem to be missing from the application.

Question 16: What did you like the least about using this product?

X

16. What did you like the least about using this product?

3 Responses

ID ↑	Name	Responses
1	anonymous	Nothing comes to my mind
2	anonymous	Signing up
3	anonymous	A

Figure 5.198: Usability Testing Results - Question 16

There were mixed responses, but one participant indicated issues with signing up. This could imply a possible area of improvement, although most other participants had no issues with the sign-up process.

Question 17: What, if anything, caused you frustration?

The image shows a digital interface for survey results. At the top right is a small 'X' icon. Below it, the question "17. What, if anything, caused you frustration?" is displayed. Underneath the question, the number "4 Responses" is shown in blue. A table follows, with columns labeled "ID ↑", "Name", and "Responses". The data is as follows:

ID ↑	Name	Responses
1	anonymous	Every thing was fine
2	anonymous	Nothing I suppose
3	anonymous	Nothing
4	anonymous	A

Figure 5.199: Usability Testing Results - Question 17

Most participants did not report any frustrations with the application, suggesting a generally positive user experience.

Please Note that the fact that not all participants answered these open-ended questions indicates that they may have found it challenging to articulate specific points, or that the questions could have been clearer.

Recommendations

Based on the findings of the usability tests, numerous suggestions for improving the STDC Web Application's usability have been formulated. These suggestions, which are based on both quantitative and qualitative data from the survey, are meant to enhance the usability of the app, boost user engagement, and boost satisfaction.

To begin, it is suggested that a desktop-friendly interface be given high priority during the future design and development of the STDC web application's new versions because that is how the vast majority of users accessed the service. The desktop is the major access point for the user base, thus while the application should stay responsive and compatible with mobile devices, it should primarily focus on maximizing the desktop user experience.

While the vast majority of users had no problems signing up, comments from a subset of users indicate that this step may use some tweaking. It could be useful to have a look over the signup procedure and see if there are any places where people get stuck or where the instructions aren't clear. In addition, a user-friendly, detailed guidance to the sign-up procedure could facilitate the first step for new customers.

While users generally had a positive experience with the app, some provided constructive criticism suggesting that more back and move forward buttons may be helpful. By giving consumers more say over how they navigate, this feature has a high potential to improve the usability of the interface.

The majority of users found the application's interface to be intuitive and simple to operate, while a few pointed out that the program should be made more user-friendly. This suggests that the user interface and usability could benefit from additional development work. The user experience might be improved through the use of current design principles, UI best practices, and elements like user-friendly iconography, consistent typography, and a clear information hierarchy.

Participants gave the app good marks for the quality of the information it supplied. However, reviews at periodic times are necessary to keep the content fresh, accu-

rate, and presented in a way that is simple to understand and absorb by readers. It also would be helpful to provide users with a 'help' or 'FAQ' page that answers frequently asked questions and provides further information on how to use the app.

User response was generally positive, and all testers felt comfortable using the application's built-in feedback system. However, the STDC may want to think about creating mechanisms to actively encourage feedback input in order to generate even higher engagement. Users can be encouraged to provide feedback by means of reminders, prompts, and even a rewards system.

Users had a generally favorable experience, giving these features high marks. However, a commitment to iteration and improvement based on user input is necessary as application development is an ongoing process. Conducting usability testing on a regular basis can assist find more problem areas and make sure the app keeps evolving to suit users' requirements and expectations. Nevertheless due to the time constraints only one round of testing could be done.

In conclusion, while the STDC online application is functional as it stands, it has room for improvement that might make it even more accessible, intuitive, and interesting to use. The STDC web application has the potential to become an even more efficient and helpful resource for its users through consistent assessment, the incorporation of user feedback, and awareness of technological advancements.

5.2.12 User Acceptance Testing

The User Acceptance Test (UAT) serves as the final phase of testing for the Student Talent Development Center (STDC) web application. This crucial step ensures that the application meets the requirements and expectations of its users. As the UAT is the last testing phase, it plays a vital role in validating the application's readiness for deployment.

Considering the project timeline and constraints, it was not possible to involve the end users or stakeholders directly in the UAT process. Therefore, this UAT will be conducted by myself. This implies that the testing will be conducted in an alpha environment.

The purpose of this document is to outline the chosen criteria for the UAT, present the test results in a structured manner, and provide a conclusion to the UAT phase.

Criteria

In selecting the criteria for the User Acceptance Test, several factors were considered. Firstly, the successful completion of previous testing phases, including inspection testing, automated unit testing, automated integration testing, build verification testing, boundary analysis testing, and usability testing, serves as an indicator of the application's stability.

Additionally, the Requirements Traceability Matrix (RTM) and Test Case tables were used as primary criteria for the UAT. The RTM ensures that all requirements have corresponding test cases, while the Test Case tables indicate the successful completion of each test case.

Furthermore, it is worth noting that the usability testing conducted for the STDC web app yielded positive feedback, please refer back to the previous sections.

UAT Results

The following table provides an overview of the requirements and their corresponding status after conducting the User Acceptance Test. The following is the column

descriptions:

- **RQ. ID:** Requirement ID.
- **Acceptance Criteria:** The specific criteria that need to be met for each requirement to be considered accepted.
- **Critical:** Indicates whether a requirement is critical or not.
- **Test Status:** The current status of each requirement based on the testing conducted.
- **UAT Result:** The result of the User Acceptance Test for each requirement, indicating whether it was accepted or not.

Table 5.11: Requirements Testing Table

RQ. ID	Acceptance Criteria	Critical	Test Status	UAT Result
RQ01	All users must be able to login using UKH email and password.	Yes	Passed	Accepted
RQ02	Tutors must be able to enter attendance for each session.	Yes	Passed	Accepted
RQ03	All users must be able to list courses taught in STDC. Current user's courses should appear on top.	Yes	Passed	Accepted
RQ04	All users must be able to view an individual course.	Yes	Passed	Accepted
RQ05	Admin must be able to create a new course.	Yes	Passed	Accepted
<i>Continued on next page</i>				

Table 5.11 – *Continued from previous page*

RQ. ID	Acceptance Criteria	Critical	Test Status	UAT Result
RQ06	Admin must be able to update a course.	Yes	Passed	Accepted
RQ07	Admin must be able to delete a course.	Yes	Passed	Accepted
RQ08	All users must be able to list sessions.	Yes	Passed	Accepted
RQ09	All users must be able to view an individual session.	Yes	Passed	Accepted
RQ10	Admin must be able to create a new session.	Yes	Passed	Accepted
RQ11	Admin must be able to update an existing session.	Yes	Passed	Accepted
RQ12	Admin must be able to delete an existing session.	Yes	Passed	Accepted
RQ13	Admin must be able to list venues.	Yes	Passed	Accepted
RQ14	All users must be able to view an individual venue.	Yes	Passed	Accepted
RQ15	Admin must be able to create (register) a new venue.	Yes	Passed	Accepted
RQ16	Admin must be able to update an existing venue.	Yes	Passed	Accepted
RQ17	Admin must be able to delete an existing venue.	Yes	Passed	Accepted
RQ18	Admin must be able to on-board tutors.	Yes	Passed	Accepted
RQ19	Admin must be able to assign tutors to courses.	Yes	Passed	Accepted
<i>Continued on next page</i>				

Table 5.11 – *Continued from previous page*

RQ. ID	Acceptance Criteria	Critical	Test Status	UAT Result
RQ20	Tutors should be able to update their profiles.	Yes	Passed	Accepted
RQ21	Admin and tutors should be able to view the attendance statistics of a student.	Yes	Passed	Accepted
RQ22	Tutors should be able to list all students in a course.	Yes	Passed	Accepted
RQ23	Tutors should be able to view the profile of a student.	Yes	Passed	Accepted
RQ24	Tutors should be able to view a student's progress and assignments.	No	Passed	Accepted
RQ25	Admin must be able to create a course schedule for each semester.	Yes	Passed	Accepted
RQ26	Admin must be able to update a course schedule.	Yes	Passed	Accepted
RQ27	Admin must be able to delete a course schedule.	Yes	Passed	Accepted
RQ28	Tutors should be able to grade students for each course.	No	Passed	Accepted
RQ29	All users should be able to access materials uploaded.	No	Passed	Accepted
<i>Continued on next page</i>				

Table 5.11 – *Continued from previous page*

RQ. ID	Acceptance Criteria	Critical	Test Status	UAT Result
RQ30	Tutors should be able to upload files (materials) for a given session of a course.	No	Passed	Accepted

Summary

In conclusion, the User Acceptance Test (UAT) for the Student Talent Development Center (STDC) web app has been conducted successfully. By leveraging the criteria set forth, including the RTM, Test Case tables, and positive usability feedback, the UAT aimed to validate the application's compliance with user expectations.

The UAT results indicate that all requirements have been tested and met, as evidenced by the successful completion of the UAT Table. This outcome provides confidence in the readiness of the STDC web app for deployment.

With the completion of the UAT, the project can now move forward to the next phase with the assurance that the application has undergone thorough testing, fulfilling the necessary quality standards.

5.3 Deployment

As the part of the software development lifecycle that is responsible for making a software system accessible to end users, the deployment phase plays an essential role in the overall process. The effective installation, setup, and functioning of the application are the key objectives of this phase, which aims to guarantee that these objectives are met. It is important to have effective oversight in place in order to guarantee accurate application deployment.

Both the backend and the frontend of the Student Talent Development Center (STDC) Web Application were given significant attention during the deployment process, which followed a two-pronged strategy. A cloud-based platform was selected for each component so that it could meet the requirements for scalability, stability, and simplicity of maintenance. During the early stages of the project, it was decided to deploy the backend, which was developed with the Ruby on Rails framework. This was done in order to make communication with the API easier. On the other hand, the frontend was not deployed until the very last step, which was when the application was finally ready for production. This strategy was founded on the reality that the API was not dependent on the frontend in any manner, but rather that the client was solely dependent on the backend.

Now that we have everything out of the way, let's take a more straightforward look at the deployment process.

5.3.1 Backend Deployment (API)

Heroku is a cloud platform as a service that supports a number of programming languages, one of which being Ruby. The backend of the STDC Web Application was developed with Ruby on Rails, and it was deployed on Heroku. When it comes to delivering web apps, Heroku offers a platform that is both reliable and expandable.

The process of deployment consisted of the following stages:

- Setting up the Ruby environment, configuring the database, and making any necessary adjustments to the configurations came first in the process of getting the application ready for deployment.
- In the second step of the process, certain environment configurations were made for Ruby on Rails apps. For example, in the production environment, static files were served. After this stage was over, the remaining Heroku-based duties were taken care of by the Heroku dashboard.
- Third, in the Heroku dashboard, the environment variables were configured, which included the establishment of secret keys and variables that were used throughout the application. This was an essential step in ensuring the confidentiality of the sensitive data. The table 5.12 is a list of environment variables that the STDC backend makes use of.

Table 5.12: Environment Variables

Environment Variable	Description
DATABASE_HOST	The database host's name
DATABASE_USERNAME	The database username
DATABASE_PASSWORD	The database password
DATABASE_PORT	The database port
SELF_URL	The application's URL
M2M_CLIENT_ID	The client ID of the machine-to-machine application call
M2M_CLIENT_SECRET	The client secret of the machine-to-machine application call
<i>Continued on next page</i>	

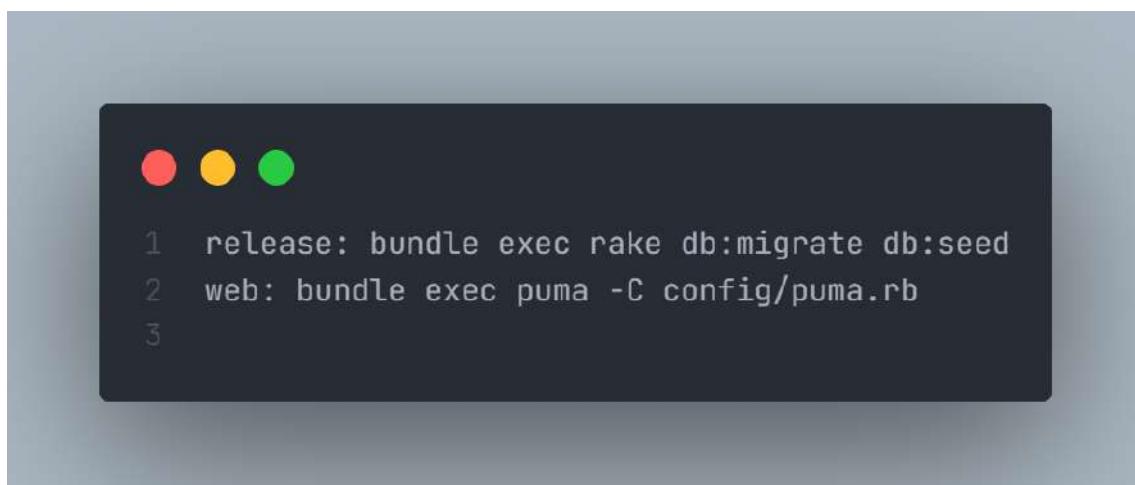
Table 5.12 – *Continued from previous page*

Environment Variable	Description
MS_URL	The URL of the user management service (AUTH0)
TOKEN_JWKS_URL	The URL of the JWKS (AUTH0) - OpenID Connect Public Keys of the Issuer
TOKEN_AUDIENCE	The audience of the token (AUTH0) - The unique identifier of the target API you want to access
USERINFO_URL	The URL of the userinfo (AUTH0) - The endpoint for user information
REDIS_URL	The URL of the Redis server
SIDEKIQ_REDIS_URL	The URL of the Redis server for Sidekiq
SIDEKIQ_SESSION_KEY	The session key of Sidekiq
GOOGLE_CLOUD_STORAGE_ACCESS_KEY	The access key of Google Cloud Storage
GOOGLE_CLOUD_STORAGE_SECRET_KEY	The secret key of Google Cloud Storage
GOOGLE_CLOUD_STORAGE_BUCKET_NAME	The bucket name of Google Cloud Storage
GOOGLE_CLOUD_STORAGE_REGION	The region of Google Cloud Storage
GOOGLE_APPLICATION_CREDENTIALS	The path of the Google application credentials
<i>Continued on next page</i>	

Table 5.12 – *Continued from previous page*

Environment Variable	Description
GOOGLE_CLOUD_STORAGE__PROJECT_ID	The project ID of Google Cloud Storage

- After that, the application was pushed to a GitHub repository, which can be accessed at <https://github.com/muhammadnawzad/stdc-api>. After that, the application was deployed to Heroku by linking the newly established Heroku application to the GitHub repository via the Heroku Command Line Interface (CLI) and the Heroku Dashboard.
- After the application was delivered, Heroku began the process by launching a web server that is capable of running the required scripts. One of these scripts is known as the Procfile, and its purpose is to detail the commands that a Heroku app will run when it first begins. A Procfile enables extra starting configuration, and while its use is not required for Heroku deployment, it is recommended.



A screenshot of a terminal window. At the top, there are three colored dots: red, yellow, and green. Below them is a dark gray rectangular area containing the following text:

```
1 release: bundle exec rake db:migrate db:seed
2 web: bundle exec puma -C config/puma.rb
3
```

Figure 5.200: Procfile

The Procfile is responsible for tailoring Heroku's default operation. Before releasing new versions, the commands "db:migrate" and "db:seed" are run with each new deployment to complete any outstanding migrations and seeds. This is done in preparation for deploying new versions. It is important to keep in mind that these instructions are disregarded if there is no need for migrations or seeds. The Puma web server, which is responsible for making the API available to users, is managed by the second section of the Procfile.

Heroku offers tools for monitoring the application, scaling it to manage greater traffic, and debugging any difficulties that may occur. The application may be viewed by going to a specific URL provided by Heroku which is the following link

<https://stdc-api.herokuapp.com>

5.3.2 Frontend Deployment (Client)

Render was used for the deployment of the frontend of the STDC Web Application, which was developed in React JS. Render is a comprehensive platform that provides an integrated solution for developing and operating applications and websites. It includes free SSL, a worldwide CDN, private networks, and automated deployments from Git.

The procedure for deployment on Render is really easy to understand. After committing the changes to the React application's GitHub repository, the application was linked to Render. Render was able to automatically recognize the Docker container that would be required to execute the React application and construct it. Render makes it possible for the client to supply any environment variables that are necessary before, during, or after the construction process. It is essential to keep in mind that React apps have the capability of consuming environment variables that are made accessible via the "REACT_APP_" prefix. The following is a list of environment variables that are utilized by the frontend.

Table 5.13: Environment Variables

Environment Variable	Description
REACT_APP_DOMAIN	The Auth0 domain
REACT_APP_CLIENT_ID	The Auth0 client ID
REACT_APP_AUDIENCE	The Auth0 audience
REACT_APP_API_URL	The URL of the API

After the build was finished, the application was deployed, and a special Render URL was made available so that users could visit it. The application can be found on

<https://stdc.onrender.com>

Render also includes tools for monitoring the program, reverting to older versions of the software, and debugging any problems that may occur.

To summarize, the deployment of the STDC Web Application demanded careful preparation and precise execution. A solution that is scalable, dependable, and easy to administer has been accomplished with the utilization of Heroku and Render for the deployment of the backend and frontend, respectively. The program may now be accessed by users, and its monitoring and maintenance can be done in a simple manner.

Chapter 6 Conclusion

This chapter is the Conclusion chapter of this thesis. Summary, Improvements, and Future Works are all integral parts of this chapter and they are as follows.

6.1 Summary

The development of this web application is a major step forward in the effort to automate the STDC's previously manual procedures at UKH. The primary objective of the project was achieved completely, and a functional and aesthetically pleasing program was developed as a result. The STDC's operations are streamlined, efficiency is increased, and communication is improved as a result of the application's automation of the workflow.

Implementing a good strategy for the development of the STDC was carefully organized and the project led to its success. However, keep in mind that developing software is an iterative process that requires regular maintenance and improvements to accommodate shifting needs and new technologies.

The importance of re-engineering routine and mundane business processes is one of the many takeaways from this work. Improvements in effectiveness, precision, and usability are just some of the outcomes that can result from automating these processes with software. This project is a great example of how technology can be used to improve upon established procedures for the good of all concerned.

Stakeholder management, technical complexity, and insufficient resources were just a few of the problems that cropped up throughout the project's duration. Overcoming these challenges called for flexibility, the ability to think carefully, and open lines of communication. In spite of all of these obstacles, the project team was able to deliver a fully operational application that meets all of the specified goals.

The knowledge and skills acquired during this project have been invaluable. Understanding of software engineering principles, project management, and stakeholder collaboration were all enhanced. Working on a real-world project like automating the

STDC's workflow allowed for practical application of theoretical concepts and best practices, which was invaluable.

In conclusion, the completion of this project is a major step toward the launch of a fully operational web application that will help automate the manual workflow currently in place at the UKH STDC. It highlights the benefits of digitizing business workflows and the iterative nature of software development. The successful completion of this project indicates the STDC's longevity and for its ongoing development and implementation of the application's features.

6.2 Future Works

There are a number of ways in which the STDC application can be enhanced in the near future. First, a more modern and aesthetically pleasing portal could help the application appeal to a wider audience and encourage more students to sign up for the STDC. Adding a "About Us," "Contact Us," and a "Frequently Asked Questions" page would be necessary to have in the STDC.

One other area of concentration is to improve the UI. The app can be made more user-friendly and aesthetically pleasing by adjusting the layout, and integrating intuitive features.

For the sake of the application's long-term maintainability, fixing the code quality issues is essential. The codebase needs to be reorganized to cut down on duplication and boost documentation. Because of this, the code will be more readable and understandable.

Since the current Windows Live login is no longer supported, it is crucial that the authentication method be modernized. A more robust and up-to-date authentication system that follows industry standards can be obtained by switching to Azure AD.

An in-app notification system will keep users aware of any developments at the STDC center. Future work on these aspects of the STDC app will allow it to grow into a more powerful and intuitive tool for users.

References

- Al-Ajlan, A. & Zedan, H., 2008. Why moodle, *in* 2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems, IEEE.
- Alvarez-Valdes, R., Crespo, E. & Tamarit, J. M., 2002. Design and implementation of a course scheduling system using tabu search, *European Journal of Operational Research*.
- Anderson, J. R., Boyle, C. F. & Reiser, B. J., 1985. Intelligent tutoring systems, *Science*.
- Åström, K. J. & Murray, R. M., 2008. Feedback systems, *Princeton Univer*.
- Carter, M. W. & Laporte, G., 1998. Recent developments in practical course timetabling, *in* international conference on the practice and theory of automated timetabling, Springer.
- Dewi, F., 2014. Edmodo: A social learning platform for blended learning class in higher education, *Research in Education Technology: Pedagogy and Technology Journal*.
- Dougiamas, M. et al., n.d.. Moodle.
- Gay, E. & Sofyan, N., 2017. The effectiveness of using edmodo in enhancing students' outcomes in advance writing course of the fifth semester at fip-ummu, *Journal of English Education*.
- Iftakhar, S., 2016. Google classroom: what works and how, *Journal of Education and Social Sciences*.
- Kc, D., 2017. Evaluation of moodle features at kajaani university of applied sciences-case study, *Procedia computer science*.
- Lan, Y.-J., Sung, Y.-t. & Chang, K.-E., 2007. A mobile-device-supported peer-assisted learning system for collaborative early efl reading, *Language learning & technology*.

Mohd Shaharanee, I. N., Jamil, J. & Mohamad Rodzi, S. S., 2016. The application of google classroom as a tool for teaching and learning, *Journal of Telecommunication, Electronic and Computer Engineering* .

Rice, W. & William, H., 2006. Moodle. birmingham.

Sadik, A., 2017. Students' acceptance of file sharing systems as a tool for sharing course materials: The case of google drive, *Education and Information Technologies* .

Sudarsana, I. K., Putra, I. B. M. A., Astawa, I. N. T. & Yogantara, I. W. L., 2019. The use of google classroom in the learning process, *in Journal of Physics: Conference Series*, IOP Publishing.

Tadelis, S., 2016. Reputation and feedback systems in online platform markets, *Annual Review of Economics* .

Topping, K. & Ehly, S., 1998. *Peer-assisted learning*, Routledge.

VanLehn, K., 2006. The behavior of tutoring systems, *International journal of artificial intelligence in education* .

VanLehn, K., 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems, *Educational psychologist* .

Wang, G., Yuan, Y., Sun, Y., Xin, J. & Zhang, Y., 2010. Peerlearning: a content-based e-learning material sharing system based on p2p network, *World Wide Web* .

Zhou, S., Xu, M. & Guan, J., 2010. Lesson: A system for lecture notes searching and sharing over internet, *Journal of Systems and Software* .

Appendices

Use Cases (Tabular Form)

System Name	Student Talent Development Center
Last Updated At	27 Dec 2022
Version	3.0

Use Case Field	Description
Use Case ID	Unique ID associated with the use case.
Version	Last updated version.
Last Updated	Date of the last modification to the use case.
Name	Name of the use case.
Traceability to	Related documents that are associated with the use case.
Actor	All users who initiate and participate in the use case.
Preconditions	Constraints that must be met for the use case to be taken.
Trigger	Event that initiates the use case.
Input	List of the data input by the user.
Steps	Steps needed to be taken to reach the use case.
Postconditions	Constraints that must be met for the use case to be done.
Exception Flow	Unsuccessful ways this use case might end.
Expected Output	The expected successful outcome of this use case.

Use Case ID	UC01		
Version	2.0	Last Updated	07 Dec 2022
Name	Login		
Traceability to	RQ01, UCM2	Actor	All Users
Preconditions	User should have a valid Microsoft account (UKH Account)		
Trigger	When user opens the application.		
Input	Valid Access Token from Microsoft.		
Steps	<ol style="list-style-type: none"> 1. Click on Login Button. 2. Sign in using Microsoft Account (UKH Account). 3. Grant permissions to the application. 		
Postconditions	<p>User is authenticated. Current User is set. User is created successfully (if not in the database). User is redirected to home page.</p>		
Exception Flow	<ol style="list-style-type: none"> 1. Error message stating “Unauthorized to sign into the application” 		
Expected Output	User is logged into the application and is the current user.		

Use Case ID	UC01.1		
Version	2.0	Last Updated	07 Dec 2022
Name	Authenticate User		
Traceability to	RQ01, UCM2	Actor	All Users
Preconditions	User must be able to get a JWT access token.		
Trigger	Login or Incoming Requests to the Server		
Input	JWT Access Token		
Steps	<ol style="list-style-type: none"> 1. Validate JWT access token. 2. Determine if the request is a machine-to-machine call. 3. Find or create user. 4. Return true. 		
Postconditions	1. User to be found or created		
Exception Flow	<ol style="list-style-type: none"> 1. Invalid JWT Signature. 2. Invalid Audience. 3. Unprocessable Entity. 		
Expected Output	For user to be successfully authenticated and found or created in the system.		

Use Case ID	UC01.2		
Version	2.0	Last Updated	07 Dec 2022
Name	Set Current User		
Traceability to	RQ01, UCM2	Actor	All Users
Preconditions	User should be authenticated		
Trigger	Authenticate User		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Determine where the request is coming from (is it m2m?) 2. If m2m, create a guest user and validate scopes. 3. else check if JWT token body is processable? 4. If it is, set the current user and return it. 5. else terminate request. 		
Postconditions	Current user variable should not to be empty.		
Exception Flow	<ol style="list-style-type: none"> 1. Invalid Scopes 3. Unprocessable Entity. 		
Expected Output	Current user is set & visible to application controllers. (Profile is visible to user in the UI).		

Use Case ID	UC02		
Version	2.0	Last Updated	07 Dec 2022
Name	Enter Attendance		
Traceability to	RQ02, UCM2	Actor	Tutor
Preconditions	Session should exist.		
Trigger	When user clicks on enter attendance button.		
Input	Students List.		
Steps	<ol style="list-style-type: none"> 1. Retrieve list of students. 2. Pick students who are present. 3. Submit Attendance. 		
Postconditions	1. List of attendees is successfully processed.		
Exception Flow	1. Unprocessable Entity.		
Expected Output	Attendance is taken and visible to users.		

Use Case ID	UC02.1					
Version	2.0	Last Updated	07 Dec 2022			
Name	List Users					
Traceability to	RQ02, UCM2, RQ18	Actor	All Users			
Preconditions	-					
Trigger	Enter Attendance or Onboard Tutor					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Send request to server 2. Process query params (process filters) 3. Wait for response 4. Send response back as an array of objects. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. An array of JSON objects or an empty array should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
List of users is returned or an empty array if no user is enrolled.						

Use Case ID	UC03					
Version	2.0	Last Updated	07 Dec 2022			
Name	List Courses					
Traceability to	RQ03, UCM2	Actor	All Users			
Preconditions	-					
Trigger	Opening Home Page or Courses					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Send request to server 2. Process query params (process filters) 3. Wait for response 4. Send response back as an array of objects. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. An array of JSON objects or an empty array should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
List of courses is returned or an empty array if no course is created.						

Use Case ID	UC04		
Version	2.0	Last Updated	07 Dec 2022
Name	View Course		
Traceability to	RQ04, UCM2	Actor	All Users
Preconditions	-		
Trigger	View Course button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Send request to server 2. Wait for response 3. Send response back as a JSON object 4. Show response back to the user. 		
Postconditions	1. A JSON object should be returned.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Course Not Found. 		
Expected Output	Course is returned and viewed with its all properties.		

Use Case ID	UC05					
Version	2.0	Last Updated	07 Dec 2022			
Name	Create Course					
Traceability to	RQ05, UCM2	Actor	Admin			
Preconditions	Course must be unique based on its ID.					
Trigger	Create Course button.					
Input	Course Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Course should be created and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Course Already Exists. 2. Unprocessable Entity. 						
Expected Output						
Course is created. Then, Course is returned and viewed with its all properties.						

Use Case ID	UC06					
Version	2.0	Last Updated	07 Dec 2022			
Name	Update Course					
Traceability to	RQ06, UCM2	Actor	Admin			
Preconditions	Course must exist.					
Trigger	Update Course button.					
Input	Course Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Course should be updated and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Course Does Not Exist. 2. Unprocessable Entity. 						
Expected Output						
Course is updated. Then, course is returned and viewed with its all properties.						

Use Case ID	UC07		
Version	2.0	Last Updated	07 Dec 2022
Name	Delete Course		
Traceability to	RQ07, UCM2	Actor	Admin
Preconditions	Course must exist.		
Trigger	Delete Course button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Click on “Delete” 2. Send request to server 3. Wait for response 4. Show response back to the user. 		
Postconditions	1. Course should be deleted in the database.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Course Does Not Exist. 		
Expected Output	Course is deleted.		

Use Case ID	UC08					
Version	3.0	Last Updated	27 Dec 2022			
Name	List Sessions					
Traceability to	RQ08, UCM2	Actor	All Users			
Preconditions	-					
Trigger	Opening Sessions Page					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Send request to server 2. Process query params (process filters) 3. Wait for response 4. Send response back as an array of objects. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. An array of JSON objects or an empty array should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
List of sessions is returned or an empty array if no session is created.						

Use Case ID	UC09		
Version	2.0	Last Updated	07 Dec 2022
Name	View Session		
Traceability to	RQ09, UCM2, AM1	Actor	All Users
Preconditions	-		
Trigger	View Session button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Send request to server 2. Wait for response 3. Send response back as a JSON object 4. Show response back to the user. 		
Postconditions	1. A JSON object should be returned.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Session Not Found. 		
Expected Output	Session is returned and viewed with its all properties.		

Use Case ID	UC10		
Version	2.0	Last Updated	07 Dec 2022
Name	Create Session		
Traceability to	RQ10, UCM2	Actor	Admin
Preconditions	Session must be unique based on its ID.		
Trigger	Create Session button.		
Input	Session Required Fields/Properties.		
Steps	<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 		
Postconditions	<ol style="list-style-type: none"> 1. Session should be created and saved into the database. 2. A JSON object should be returned. 		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Session Already Exists. 2. Unprocessable Entity. 		
Expected Output	Session is created. Then, session is returned and viewed with its all properties.		

Use Case ID	UC11		
Version	2.0	Last Updated	07 Dec 2022
Name	Update Session		
Traceability to	RQ11, UCM2	Actor	Admin
Preconditions	Session must exist.		
Trigger	Update Session button.		
Input	Session Required Fields/Properties.		
Steps	<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 		
Postconditions	<ol style="list-style-type: none"> 1. Session should be updated and saved into the database. 2. A JSON object should be returned. 		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Session Does Not Exist. 2. Unprocessable Entity. 		
Expected Output	Session is updated. Then, session is returned and viewed with its all properties.		

Use Case ID	UC12		
Version	2.0	Last Updated	07 Dec 2022
Name	Delete Session		
Traceability to	RQ12, UCM2	Actor	Admin
Preconditions	Session must exist.		
Trigger	Delete Session button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Click on “Delete” 2. Send request to server 3. Wait for response 4. Show response back to the user. 		
Postconditions	1. Session should be deleted in the database.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Session Does Not Exist. 		
Expected Output	Session is deleted.		

Use Case ID	UC13					
Version	2.0	Last Updated	07 Dec 2022			
Name	List Venues					
Traceability to	RQ13, UCM2	Actor	All Users			
Preconditions	-					
Trigger	Opening Home Page or Venues.					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Send request to server 2. Process query params (process filters) 3. Wait for response 4. Send response back as an array of objects. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. An array of JSON objects or an empty array should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
List of venues is returned or an empty array if no venue is created.						

Use Case ID	UC14		
Version	2.0	Last Updated	07 Dec 2022
Name	View Venue		
Traceability to	RQ14, UCM2	Actor	All Users
Preconditions	-		
Trigger	View Venue button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Send request to server 2. Wait for response 3. Send response back as a JSON object 4. Show response back to the user. 		
Postconditions	1. A JSON object should be returned.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Venue Not Found. 		
Expected Output	Venue is returned and viewed with its all properties.		

Use Case ID	UC15					
Version	2.0	Last Updated	07 Dec 2022			
Name	Create Venue					
Traceability to	RQ15, UCM2	Actor	Admin			
Preconditions	Venue must be unique based on its ID.					
Trigger	Create Venue button.					
Input	Venue Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Venue should be created and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Venue Already Exists. 2. Unprocessable Entity. 						
Expected Output						
Venue is created. Then, venue is returned and viewed with its all properties.						

Use Case ID	UC16					
Version	2.0	Last Updated	07 Dec 2022			
Name	Update Venue					
Traceability to	RQ16, UCM2	Actor	Admin			
Preconditions	Venue must exist.					
Trigger	Update Venue button.					
Input	Venue Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Venue should be updated and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Venue Does Not Exist. 2. Unprocessable Entity. 						
Expected Output						
Venue is updated. Then, venue is returned and viewed with its all properties.						

Use Case ID	UC17		
Version	2.0	Last Updated	07 Dec 2022
Name	Delete Venue		
Traceability to	RQ17, UCM2	Actor	Admin
Preconditions	Venue must exist.		
Trigger	Delete Venue button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Click on “Delete” 2. Send request to server 3. Wait for response 4. Show response back to the user. 		
Postconditions	1. Venue should be deleted in the database.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Venue Does Not Exist. 		
Expected Output	Venue is deleted.		

Use Case ID	UC18					
Version	2.0	Last Updated	07 Dec 2022			
Name	Onboard Tutor					
Traceability to	RQ18, UCM2	Actor	Admin			
Preconditions	Tutor must be enrolled in the system as a student first.					
Trigger	Edit button					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Go to users 2. Search for user. 3. Wait for response 4. Edit user's role and set it as "Tutor". 5. Submit Request. 						
Postconditions						
<ol style="list-style-type: none"> 1. Updating user roles successfully. 2. Response is sent back and shown to user. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 						
Expected Output						
For user to be onboarded as a tutor. ("tutor" role is assigned to the user).						

Use Case ID	UC19					
Version	2.0	Last Updated	07 Dec 2022			
Name	Assign Tutor					
Traceability to	RQ19, UCM2	Actor	Admin			
Preconditions	Course must exist.					
Trigger	Assign Tutor button					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Go to courses 2. Search for course 3. Click on “Assign Course” 4. Search for tutor and then select tutor. 5. Submit Request. 						
Postconditions						
<ol style="list-style-type: none"> 1. Updating course successfully. 2. Response is sent back and shown to user. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 						
Expected Output						
Course to be assigned to a tutor.						

Use Case ID	UC20		
Version	2.0	Last Updated	07 Dec 2022
Name	View Attendance		
Traceability to	RQ20, UCM2	Actor	Admin Tutor
Preconditions	Session must exist.		
Trigger	View Attendance button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. View Session 2. Click on “View Attendance” 3. Send request to the server 4. Fetch response and render the list. 5. Show response back to the user. 		
Postconditions	1. An array of JSON objects should be returned or an empty array in case of none.		
Exception Flow	1. Unauthorized Access.		
Expected Output	Attendance list is returned and shown to the user.		

Use Case ID	UC21					
Version	2.0	Last Updated	07 Dec 2022			
Name	View Time and Venue					
Traceability to	RQ21, UCM3	Actor	All Users			
Preconditions	Session must exist.					
Trigger	View Session					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. View Session 2. Send request to the server 3. Wait for data to be fetched 4. Fetch response and render the list. 5. Show response back to the user. 						
Postconditions						
-						
Exception Flow						
-						
Expected Output						
Time and venue of the session is returned and shown to the user.						

Use Case ID	UC22					
Version	2.0	Last Updated	07 Dec 2022			
Name	Assign Time and Venue					
Traceability to	RQ22, UCM3	Actor	Admin			
Preconditions	Venue shouldn't be occupied.					
Trigger	Assign Time and Venue button					
Input	Time, Venue.					
Steps						
<ol style="list-style-type: none"> 1. Go to sessions 2. Search for session 3. Click on "Assign Time and Venue" 4. Select time and venue. 5. Submit Request. 						
Postconditions						
<ol style="list-style-type: none"> 1. Updating session successfully. 2. Updating venue status to occupied. 3. Response is sent back and shown to user. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 						
Expected Output						
Time and venue should be assigned to a session.						

Use Case ID	UC23		
Version	3.0	Last Updated	27 Dec 2022
Name	View History		
Traceability to	RQ23, UCM3	Actor	Student Tutor
Preconditions	-		
Trigger	View History button		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Go to sessions 2. Click on “View Session History” 3. Wait for response 4. View session history 		
Postconditions	<ol style="list-style-type: none"> 1. Response is sent back and shown to user. 		
Exception Flow	<ul style="list-style-type: none"> - 		
Expected Output	List of sessions attended will be returned and shown to the user.		

Use Case ID	UC24					
Version	2.0	Last Updated	07 Dec 2022			
Name	Enter Availability					
Traceability to	RQ24, UCM3	Actor	Tutor			
Preconditions	-					
Trigger	Enter Availability button.					
Input	Available time slots.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. User available times should be updated. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 						
Expected Output						
Tutor availability is set and visible to the others.						

Use Case ID	UC25		
Version	2.0	Last Updated	07 Dec 2022
Name	Cancel Session		
Traceability to	RQ25, UCM3	Actor	Tutor
Preconditions	-		
Trigger	Cancel Session button		
Input	-		
Steps	<ol style="list-style-type: none"> 1. View Session 2. Click on “Cancel Session” 3. Wait for response 4. View changes in session status. 		
Postconditions	<ol style="list-style-type: none"> 1. Response is sent back and shown to user. 		
Exception Flow	<ul style="list-style-type: none"> - 		
Expected Output	Session is cancelled (Session status is cancelled) and visible to the users.		

Use Case ID	UC26					
Version	2.0	Last Updated	07 Dec 2022			
Name	View Hours					
Traceability to	RQ26, UCM3	Actor	Tutor Admin			
Preconditions	-					
Trigger	User Profile Page					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. View Session 2. Send request to the server 3. Wait for data to be fetched 4. Fetch response and render the response. 5. Show response back to the user. 						
Postconditions						
-						
Exception Flow						
-						
Expected Output						
Summation of tutored hours is returned and shown to the user.						

Use Case ID	UC27					
Version	2.0	Last Updated	07 Dec 2022			
Name	Provide Feedback					
Traceability to	RQ27, UCM4, AM2	Actor	Student			
Preconditions	-					
Trigger	Provide Feedback button.					
Input	Feedback Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. View Session. 2. Click on Provide Feedback button and fill the required fields. 3. Send request to server 4. Wait for response 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Feedback should be created and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 						
Expected Output						
Feedback is provided. Then, feedback is returned and viewed with its all properties.						

Use Case ID	UC28					
Version	2.0	Last Updated	07 Dec 2022			
Name	View Feedbacks					
Traceability to	RQ28, UCM4	Actor	Admin			
Preconditions	-					
Trigger	Feedbacks section.					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Open sidebar. 2. Click on “Feedbacks” 3. Wait for data to be fetched 4. Fetch response and render the response. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. List of feedbacks should be fetched. 						
Exception Flow						
-						
Expected Output						
List of students' feedbacks is returned and shown to the Admin.						

Use Case ID	UC29		
Version	3.0	Last Updated	27 Dec 2022
Name	Access Materials		
Traceability to	RQ29, UCM5	Actor	All Users
Preconditions	Material should exist.		
Trigger	View Session		
Input	-		
Steps	<ol style="list-style-type: none"> 1. View Session 2. Wait for data to be fetched 3. Fetch list of materials. 4. Show list of materials back to the user and let them download them. 		
Postconditions	1. List of materials should be fetched.		
Exception Flow	-		
Expected Output	List of materials is returned and shown to the users and accessible to be downloaded.		

Use Case ID	UC30		
Version	2.0	Last Updated	07 Dec 2022
Name	Upload Material		
Traceability to	RQ30, UCM5, AM3	Actor	Tutor
Preconditions	File must not be larger than 5 MB.		
Trigger	“Upload Material” button when “View Session”		
Input	Upload Material Required Fields/Properties.		
Steps	<ol style="list-style-type: none"> 1. View Session. 2. Click on Upload Material button and fill the required fields. 3. Send request to server 4. Wait for response 5. Show response back to the user. 		
Postconditions	<ol style="list-style-type: none"> 1. File should be uploaded and saved into the database. 2. File should be sanitized. 		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Unprocessable Entity. 		
Expected Output	Material is uploaded into the system and visible to all users.		

Use Case ID	UC31					
Version	1.0	Last Updated	27 Dec 2022			
Name	View Feedback					
Traceability to	RQ31, UCM4	Actor	Admin			
Preconditions	Feedback must exist.					
Trigger	“View Feedback” button when all feedbacks are viewed					
Input	-.					
Steps						
<ol style="list-style-type: none"> 1. View Feedbacks. 2. Click on View Feedback Button. 3. Send request to server 4. Wait for response 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Feedback should be fetched and rendered properly. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
Feedback details are visible and rendered to the Admin.						

Use Case ID	UC32					
Version	1.0	Last Updated	27 Dec 2022			
Name	View Profile					
Traceability to	RQ32, UCM2	Actor	All Users			
Preconditions	User should exist and be logged in.					
Trigger	“Profile” button on the sidebar					
Input	-.					
Steps						
<ol style="list-style-type: none"> 1. Login 2. Open sidebar. 3. Click on Profile. 4. Wait for response 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. User information should be fetched and rendered properly. 						
Exception Flow						
<p>-</p>						
Expected Output						
User profile details are visible and rendered to themselves.						

Use Case ID	UC33					
Version	1.0	Last Updated	27 Dec 2022			
Name	Register for Session					
Traceability to	RQ33, UCM3	Actor	Student			
Preconditions	Session should not be expired.					
Trigger	“Register” button when session is viewed					
Input	User details.					
Steps						
<ol style="list-style-type: none"> 1. Login 2. View Session. 3. Click on Register. 4. Provide required information. 5. Submit Request. 						
Postconditions						
-						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized access. 2. Unable to register (Session Expired). 						
Expected Output						
User registers for upcoming session.						

Use Case ID	UC34					
Version	1.0	Last Updated	27 Dec 2022			
Name	View Registered Students					
Traceability to	RQ34, UCM3	Actor	Tutor			
Preconditions	-					
Trigger	“View Registered Students” button when session is viewed					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Login 2. View Session. 3. Click on View Registered Students. 4. Wait for response. 5. Show response back to the user. 						
Postconditions						
-						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized access. 						
Expected Output						
User can view students who have registered for the upcoming session.						

Use Case ID	UC35					
Version	1.0	Last Updated	14 June 2023			
Name	List Questions					
Traceability to	RQ35, UCM4	Actor	All Users			
Preconditions	-					
Trigger	Opening Home Page or Questions.					
Input	-					
Steps						
<ol style="list-style-type: none"> 1. Send request to server 2. Process query params (process filters) 3. Wait for response 4. Send response back as an array of objects. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. An array of JSON objects or an empty array should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 						
Expected Output						
List of questions is returned or an empty array if no question is created.						

Use Case ID	UC36		
Version	1.0	Last Updated	14 June 2023
Name	View Question		
Traceability to	RQ36, UCM4	Actor	All Users
Preconditions	-		
Trigger	View Question button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Send request to server 2. Wait for response 3. Send response back as a JSON object 4. Show response back to the user. 		
Postconditions	1. A JSON object should be returned.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Question Not Found. 		
Expected Output	Question is returned and viewed with its all properties.		

Use Case ID	UC37					
Version	1.0	Last Updated	14 June 2023			
Name	Create Question					
Traceability to	RQ37, UCM4	Actor	Admin			
Preconditions	Question must be unique based on its ID.					
Trigger	Create Question button.					
Input	Question Required Fields/Properties.					
Steps						
<ol style="list-style-type: none"> 1. Fill the required fields. 2. Send request to server 3. Wait for response 4. Send response back as a JSON object. 5. Show response back to the user. 						
Postconditions						
<ol style="list-style-type: none"> 1. Question should be created and saved into the database. 2. A JSON object should be returned. 						
Exception Flow						
<ol style="list-style-type: none"> 1. Unauthorized Access. 3. Question Already Exists. 2. Unprocessable Entity. 						
Expected Output						
Question is created. Then, question is returned and viewed with its all properties.						

Use Case ID	UC38		
Version	1.0	Last Updated	14 June 2023
Name	Delete Question		
Traceability to	RQ38, UCM4	Actor	Admin
Preconditions	Question must exist.		
Trigger	Delete Question button.		
Input	-		
Steps	<ol style="list-style-type: none"> 1. Click on “Delete” 2. Send request to server 3. Wait for response 4. Show response back to the user. 		
Postconditions	1. Question should be deleted in the database.		
Exception Flow	<ol style="list-style-type: none"> 1. Unauthorized Access. 2. Question Does Not Exist. 		
Expected Output	Question is deleted.		

Basic Functionalities

All Users

- [M] All users must be able to login.
User must use UKH email and Password.
- [M] All users must see how many modules are taught in STDC.
User must able to view them as a list.
 - [S] Student and Tutor should see how many of them they are enrolled in.
User sees enrolled modules first when listing modules.

Tutor

- [S] Tutor must be able to enter attendance of each session.

Admin

- [M] Admin must be able to view, create, update, and delete modules and sessions.
They also do the same for Venue. Also, sperate each task into a different requirement.
- [M] Admin must be able to onboard tutors.
- [M] Admin must be able to assign tutors modules.
- [S] Admin must be able to see the list of attendance of each session.
When viewing session, there must be a button.

Scheduling Component

All Users

- [M] All users must see upcoming lesson's time and venue
Shown when viewing session.
 - [S] Student and Tutor should see the history of sessions they have attended.
- [W] All users could request to schedule extra sessions.

Student

- [W] Student could register for upcoming sessions.

Tutor

- [S] Tutor should be able to enter their availability.
- [C] Tutor could cancel a session.
- [C] Tutor could see how many hours they have tutored.
- [W] Tutor could view requested extra sessions from students and approve of them.

Admin

- [M] Admin must be able to assign time and venue to each session.
- [C] Admin could see how many hours each tutor has tutored.

When viewing their profile.

- [W] Admin could see requested sessions & session cancellations and approve them.
- [W] Admin could send email to notify tutors and students through the system.

Feedback Component

Student

- [S] Student should be able to provide feedback after each session.

There should be a button to indicate “Provide Feedback”.

Admin

- [S] Admin should be able to see student feedback.
- [W] Admin could take Annual Survey.

Resource Component

Student

- [S] Student should access the material uploaded for each module grouped by sessions.

Files (Materials) are uploaded to session, not module.

Tutor

- [S] Tutors should be able to upload material for a given session of a module.

Some Requirements are repetition of previous ones, such as “view hours tutored”, and
some requirements are very generalized such as “manage module and sessions”.

1. Keep requirements consistent.
2. Be more specific with requirements.
3. Change Basic functionalities to “System Core”
4. IDs are all over the place, keep ids sequential.

These requirements need to be updated.

Test Cases

The Followings are generated test cases for “Student Talent Development Center Web Application”. These test cases are tentative to change with requirement changes. Requirement associated with each test case can be found in the “Requirements Traceability Matrix”.

This document is last Updated in 22/06/2023.

System Name	Student Talent Development Center
Pending	0
Passed	114
Failed	0

Test Case ID	TC01		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	All users must be able to login using UKH email and Password.		
Expected Output	User is logged into the application and is the current user.		

Test Case ID	TC02		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Tutor must be able to enter attendance of each session.</p>		
Expected Output	<p>Attendance is taken and visible to users.</p>		

Test Case ID	TC03		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>All users must list courses taught in STDC. (Current User's courses should appear on top).</p>		
Expected Output	<p>List of courses is returned or an empty array if no course is created.</p>		

Test Case ID	TC04		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
All users must be able to view an individual course.			
Expected Output			
Course is returned and viewed with its all properties.			

Test Case ID	TC05		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to create a new course.			
Expected Output			
Course is created. Then, course is returned and viewed with its all properties.			

Test Case ID	TC06		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to update a course.</p>		
Expected Output	<p>Course is updated. Then, course is returned and viewed with its all properties.</p>		

Test Case ID	TC07		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to delete a course.</p>		
Expected Output	<p>Course is deleted.</p>		

Test Case ID	TC08		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
All users must be able to list sessions.			
Expected Output			
List of sessions is returned or an empty array if no session is created.			

Test Case ID	TC09		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
All users must be able to view an individual session.			
Expected Output			
Session is returned and viewed with its all properties.			

Test Case ID	TC10		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to create a new session.</p>		
Expected Output	<p>Session is created. Then, session is returned and viewed with its all properties.</p>		

Test Case ID	TC11		
Last Test Date	10/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to update an existing session.</p>		
Expected Output	<p>Session is updated. Then, session is returned and viewed with its all properties.</p>		

Test Case ID	TC12		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to delete an existing session.			
Expected Output			
Session is deleted.			

Test Case ID	TC13		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to list venues.			
Expected Output			
List of venues is returned or an empty array if no venue is created.			

Test Case ID	TC14		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
All users must be able to view an individual venue.			
Expected Output			
Venue is returned and viewed with its all properties.			

Test Case ID	TC15		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to create (Register) a new venue.			
Expected Output			
Venue is created. Then, venue is returned and viewed with its all properties.			

Test Case ID	TC16		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to update an existing venue.			
Expected Output			
Venue is updated. Then, venue is returned and viewed with its all properties.			

Test Case ID	TC17		
Last Test Date	24/05/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to delete an existing venue.			
Expected Output			
Venue is deleted.			

Test Case ID	TC18		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
	Admin must be able to onboards tutors.		
Expected Output			
	For user to be onboarded as a tutor. ("tutor" role is assigned to the user).		

Test Case ID	TC19		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
	Admin must be able to assign tutors courses.		
Expected Output			
	Course to be assigned to a tutor.		

Test Case ID	TC20		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin and Tutor should be able to list the attendance of a session using specified button.</p>		
Expected Output	<p>Attendance list is returned and shown to the user.</p>		

Test Case ID	TC21		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>All users must see upcoming lesson's time and venue when they view a session.</p>		
Expected Output	<p>Time and venue of the session is returned and shown to the user.</p>		

Test Case ID	TC22		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to assign time and venue to each session.</p>		
Expected Output	<p>Time and venue should be assigned to a session.</p>		

Test Case ID	TC23		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Student and Tutor should see the history of sessions they have attended.</p>		
Expected Output	<p>List of sessions attended will be returned and shown to the user.</p>		

Test Case ID	TC24		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Tutor should be able to enter their availability.</p>		
Expected Output	<p>Tutor availability is set and visible to the others.</p>		

Test Case ID	TC25		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Tutor could cancel a session.</p>		
Expected Output	<p>Session is cancelled (Session status is cancelled) and visible to the users.</p>		

Test Case ID	TC26		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin and Tutor could see how many hours tutor have tutored when viewing tutor profile.</p>		
Expected Output	<p>Summation of tutored hours is returned and shown to the user.</p>		

Test Case ID	TC27		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Student should be able to provide feedback after each session.</p>		
Expected Output	<p>Feedback is provided. Then, feedback is returned and viewed with its all properties.</p>		

Test Case ID	TC28		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin should be able to list student feedbacks.</p>		
Expected Output	<p>List of students' feedbacks is returned and shown to the Admin.</p>		

Test Case ID	TC29		
Last Test Date	02/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>All users should be able to access materials uploaded.</p>		
Expected Output	<p>List of materials is returned and shown to the users and accessible to be downloaded.</p>		

Test Case ID	TC30		
Last Test Date	12/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Tutors should be able to upload files (material) for a given session of a course.			
Expected Output			
Material is uploaded into the system and visible to all users.			

Test Case ID	TC31		
Last Test Date	12/06/2023	Last Updated At	27 Dec 2022
Status	Passed		
Description			
Admin should be able to view student feedback individually.			
Expected Output			
Feedback details are visible and rendered to the Admin.			

Test Case ID	TC32		
Last Test Date	12/06/2023	Last Updated At	27 Dec 2022
Status	Passed		
Description	<p>All users should be able to view their profile and its details.</p>		

Test Case ID	TC33		
Last Test Date	15/06/2023	Last Updated At	27 Dec 2022
Status	Passed		
Description	<p>Students should be able to register for upcoming session.</p>		
Expected Output	<p>Student registers for upcoming session.</p>		

Test Case ID	TC34		
Last Test Date	15/06/2023	Last Updated At	27 Dec 2022
Status	Passed		
Description	<p>Tutor should be able to view students who have registered for upcoming session.</p>		
Expected Output	<p>User can view students who have registered for the upcoming session.</p>		

Test Case ID	TC35		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description	<p>Admin must be able to list questions.</p>		
Expected Output	<p>List of questions is returned or an empty array if no question is created.</p>		

Test Case ID	TC36		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
All users must be able to view an individual question.			
Expected Output			
Question is returned and viewed with its all properties.			

Test Case ID	TC37		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to create (Register) a new question.			
Expected Output			
Question is created. Then, question is returned and viewed with its all properties.			

Test Case ID	TC38		
Last Test Date	15/06/2023	Last Updated At	15 June 2023
Status	Passed		
Description			
Admin must be able to delete an existing question.			
Expected Output			
Question is deleted.			

Test Case ID	TC39		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that an Answer object with valid attributes is considered valid.			
Expected Output			
The Answer object should be valid.			

Test Case ID	TC40		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Answer model only accepts the values 'active' or 'inactive'.			
Expected Output			
An Answer object with a 'status' value other than 'active' or 'inactive' should be considered invalid.			

Test Case ID	TC41		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that an Answer object must have a non-empty 'content' attribute.			
Expected Output			
An Answer object with an empty 'content' value should be considered invalid.			

Test Case ID	TC42		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that an Answer object must be associated with a Feedback object.			
Expected Output			
An Answer object without a associated Feedback should be considered invalid.			

Test Case ID	TC43		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that an Answer object must be associated with a Question object.			
Expected Output			
An Answer object without a associated Question should be considered invalid.			

Test Case ID	TC44		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that an Attendance object with valid attributes is considered valid.			
Expected Output			
The Attendance object should be valid.			

Test Case ID	TC45		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that the 'rating' attribute of the Attendance model only accepts values between 1 and 5.			
Expected Output			
An Attendance object with a 'rating' value outside the range of 1 to 5 should be considered invalid.			

Test Case ID	TC46		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that the 'registration' attribute of the Attendance model only accepts the values 'registered' or 'unregistered'.			
Expected Output			
An Attendance object with a 'registration' value other than 'registered' or 'unregistered' should be considered invalid.			

Test Case ID	TC47		
Last Test Date	16/06/2023	Last Updated At	16 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Attendance model only accepts the values 'attended' or 'unattended'.			
Expected Output			
An Attendance object with a 'status' value other than 'attended' or 'unattended' should be considered invalid.			

Test Case ID	TC48		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that an Attendance object must be associated with a User object.			
Expected Output			
An Attendance object without an associated User should be considered invalid.			

Test Case ID	TC49		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that an Attendance object must be associated with a Session object.			
Expected Output			
An Attendance object without an associated Session should be considered invalid.			

Test Case ID	TC50		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that a Course object with valid attributes is considered valid.			
Expected Output			
The Course object should be valid.			

Test Case ID	TC51		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that a Course object must have a non-empty 'name' attribute.			
Expected Output			
A Course object with an empty 'name' value should be considered invalid.			

Test Case ID	TC52		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Course model only accepts the values 'active' or 'inactive'.			
Expected Output			
A Course object with a 'status' value other than 'active' or 'inactive' should be considered invalid.			

Test Case ID	TC53		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that a Course object can have multiple enrollments.			
Expected Output			
The Course object should be associated with multiple Enrollment objects.			

Test Case ID	TC54		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description	<p>Verify that a Course object can have multiple users enrolled through enrollments.</p>		

Test Case ID	TC55		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description	<p>Verify that an Enrollment object with valid attributes is considered valid.</p>		
Expected Output	<p>The Enrollment object should be valid.</p>		

Test Case ID	TC56		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that an Enrollment object must be associated with a User object.			
Expected Output			
An Enrollment object without an associated User should be considered invalid.			

Test Case ID	TC57		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that an Enrollment object must be associated with a Course object.			
Expected Output			
An Enrollment object without an associated Course should be considered invalid.			

Test Case ID	TC58		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that a Feedback object with valid attributes is considered valid.			
Expected Output			
The Feedback object should be valid.			

Test Case ID	TC59		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description			
Verify that the 'rating' attribute of the Feedback model only accepts values between 1 and 5.			
Expected Output			
A Feedback object with a 'rating' value outside the range of 1 to 5 should be considered invalid.			

Test Case ID	TC60		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description	<p>Verify that the 'state' attribute of the Feedback model only accepts the values 'read' or 'unread'.</p>		
Expected Output	<p>A Feedback object with a 'state' value other than 'read' or 'unread' should be considered invalid.</p>		

Test Case ID	TC61		
Last Test Date	17/06/2023	Last Updated At	17 June 2023
Status	Passed		
Description	<p>Verify that a Feedback object must be associated with a User object.</p>		
Expected Output	<p>A Feedback object without an associated User should be considered invalid.</p>		

Test Case ID	TC62		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Feedback object must be associated with a Session object.			
Expected Output			
A Feedback object without an associated Session should be considered invalid.			

Test Case ID	TC63		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Feedback object can have multiple answers.			
Expected Output			
The Feedback object should be associated with multiple Answer objects.			

Test Case ID	TC64		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Feedback object can have multiple questions through answers.			
Expected Output			
The Feedback object should be associated with multiple Question objects through answers.			

Test Case ID	TC65		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Question object with valid attributes is considered valid.			
Expected Output			
The Question object should be valid.			

Test Case ID	TC66		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Question model only accepts the values 'active' or 'inactive'.			
Expected Output			
A Question object with a 'status' value other than 'active' or 'inactive' should be considered invalid.			

Test Case ID	TC67		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Question object must have a non-empty 'title' attribute.			
Expected Output			
A Question object with an empty 'title' value should be considered invalid.			

Test Case ID	TC68		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description	<p>Verify that a Question object can have multiple answers.</p>		
Expected Output	<p>The Question object should be associated with multiple Answer objects.</p>		

Test Case ID	TC69		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description	<p>Verify that a Question object can have multiple feedbacks through answers.</p>		
Expected Output	<p>The Question object should be associated with multiple Feedback objects through answers.</p>		

Test Case ID	TC70		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Material object with valid attributes is considered valid.			
Expected Output			
The Material object should be valid.			

Test Case ID	TC71		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'key' attribute.			
Expected Output			
A Material object with an empty 'key' value should be considered invalid.			

Test Case ID	TC72		
Last Test Date	18/06/2023	Last Updated At	18 June 2023
Status	Passed		
Description			
Verify that a Material object must have a unique 'key' attribute.			
Expected Output			
A Material object with a 'key' value that is already used by another Material object should be considered invalid.			

Test Case ID	TC73		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'bucket' attribute.			
Expected Output			
A Material object with an empty 'bucket' value should be considered invalid.			

Test Case ID	TC74		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'content_type' attribute.			
Expected Output			
A Material object with an empty 'content_type' value should be considered invalid.			

Test Case ID	TC75		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'name' attribute.			
Expected Output			
A Material object with an empty 'name' value should be considered invalid.			

Test Case ID	TC76		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'url' attribute.			
Expected Output			
A Material object with an empty 'url' value should be considered invalid.			

Test Case ID	TC77		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that a Material object must have a non-empty 'size' attribute.			
Expected Output			
A Material object with an empty 'size' value should be considered invalid.			

Test Case ID	TC78		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that the 'service' attribute of the Material model only accepts the value 'gcs'.			
Expected Output			
A Material object with a 'service' value other than 'gcs' should be considered invalid.			

Test Case ID	TC79		
Last Test Date	19/06/2023	Last Updated At	19 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Material model only accepts the values 'active' or 'inactive'.			
Expected Output			
A Material object with a 'status' value other than 'active' or 'inactive' should be considered invalid.			

Test Case ID	TC80		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that the 'visibility' attribute of the Material model only accepts the values 'visible' or 'invisible'.			
Expected Output			
A Material object with a 'visibility' value other than 'visible' or 'invisible' should be considered invalid.			

Test Case ID	TC81		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Material object must be associated with a User object.			
Expected Output			
A Material object without an associated User should be considered invalid.			

Test Case ID	TC82		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Material object must be associated with a Session object.			
Expected Output			
A Material object without an associated Session should be considered invalid.			

Test Case ID	TC83		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Venue object with valid attributes is considered valid.			
Expected Output			
The Venue object should be valid.			

Test Case ID	TC84		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Venue object must have a non-empty 'name' attribute.			
Expected Output			
A Venue object with an empty 'name' value should be considered invalid.			

Test Case ID	TC85		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Venue object must have a non-empty 'capacity' attribute.			
Expected Output			
A Venue object with an empty 'capacity' value should be considered invalid.			

Test Case ID	TC86		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description	<p>Verify that the 'status' attribute of the Venue model only accepts the values 'active' or 'inactive'.</p>		
Expected Output	<p>A Venue object with a 'status' value other than 'active' or 'inactive' should be considered invalid.</p>		

Test Case ID	TC87		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description	<p>Verify that the 'floor' attribute of the Venue model only accepts specific values.</p>		
Expected Output	<p>A Venue object with a 'floor' value that is not one of the defined floor options should be considered invalid.</p>		

Test Case ID	TC88		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that the 'availability' attribute of the Venue model only accepts the values 'available' or 'reserved'.			
Expected Output			
A Venue object with an 'availability' value other than 'available' or 'reserved' should be considered invalid.			

Test Case ID	TC89		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a Venue object can have multiple sessions.			
Expected Output			
The Venue object should be associated with multiple Session objects.			

Test Case ID	TC90		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object with valid attributes is considered valid.			
Expected Output			
The User object should be valid.			

Test Case ID	TC91		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the User model only accepts the values 'inactive' or 'active'.			
Expected Output			
A User object with a 'status' value other than 'inactive' or 'active' should be considered invalid.			

Test Case ID	TC92		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a non-empty 'remote_id' attribute.			
Expected Output			
A User object with an empty 'remote_id' value should be considered invalid.			

Test Case ID	TC93		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a unique 'remote_id' attribute.			
Expected Output			
A User object with a 'remote_id' value that is already used by another User object should be considered invalid.			

Test Case ID	TC94		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a non-empty 'email' attribute.			
Expected Output			
A User object with an empty 'email' value should be considered invalid.			

Test Case ID	TC95		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a unique 'email' attribute.			
Expected Output			
A User object with an 'email' value that is already used by another User object should be considered invalid.			

Test Case ID	TC96		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a non-empty 'first_name' attribute.			
Expected Output			
A User object with an empty 'first_name' value should be considered invalid.			

Test Case ID	TC97		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a non-empty 'last_name' attribute.			
Expected Output			
A User object with an empty 'last_name' value should be considered invalid.			

Test Case ID	TC98		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object must have a non-empty 'name' attribute.			
Expected Output			
A User object with an empty 'name' value should be considered invalid.			

Test Case ID	TC99		
Last Test Date	20/06/2023	Last Updated At	20 June 2023
Status	Passed		
Description			
Verify that a User object can have multiple enrollments.			
Expected Output			
The User object should be associated with multiple Enrollment objects.			

Test Case ID	TC100		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description	<p>Verify that a User object can have multiple courses through enrollments.</p>		

Test Case ID	TC101		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description	<p>Verify that a User object can have multiple attendances.</p>		
Expected Output	<p>The User object should be associated with multiple Course objects through enrollments.</p>		

Test Case ID	TC102		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a User object can have multiple sessions through attendances.			
Expected Output			
The User object should be associated with multiple Session objects through attendances.			

Test Case ID	TC103		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object with valid attributes is considered valid.			
Expected Output			
The Session object should be valid.			

Test Case ID	TC104		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object must have a non-empty 'name' attribute.			
Expected Output			
A Session object with an empty 'name' value should be considered invalid.			

Test Case ID	TC105		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object must have a non-empty 'start_time' attribute.			
Expected Output			
A Session object with an empty 'start_time' value should be considered invalid.			

Test Case ID	TC106		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object must have a non-empty 'end_time' attribute.			
Expected Output			
A Session object with an empty 'end_time' value should be considered invalid.			

Test Case ID	TC107		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object must have a non-empty 'date' attribute.			
Expected Output			
A Session object with an empty 'date' value should be considered invalid.			

Test Case ID	TC108		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object must have a non-empty 'duration' attribute.			
Expected Output			
A Session object with an empty 'duration' value should be considered invalid.			

Test Case ID	TC109		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that the 'status' attribute of the Session model only accepts the values 'active' or 'inactive'.			
Expected Output			
A Session object with a 'status' value other than 'active' or 'inactive' should be considered invalid.			

Test Case ID	TC110		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
	Verify that the 'cancellation' attribute of the Session model only accepts specific values.		
Expected Output			
	A Session object with a 'cancellation' value that is not one of the defined cancellation options should be considered invalid.		

Test Case ID	TC111		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
	Verify that a Session object belongs to a Course.		
Expected Output			
	The Session object should be associated with a Course object.		

Test Case ID	TC112		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object belongs to a Venue.			
Expected Output			
The Session object should be associated with a Venue object.			

Test Case ID	TC113		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description			
Verify that a Session object can have multiple attendances.			
Expected Output			
The Session object should be associated with multiple Attendance objects.			

Test Case ID	TC114		
Last Test Date	21/06/2023	Last Updated At	21 June 2023
Status	Passed		
Description	<p>Verify that a Session object can have multiple users through attendances.</p>		
Expected Output	<p>The Session object should be associated with multiple User objects through attendances.</p>		