JMJ MARIST BROTHERSMIT 261 - Project
Notre Dame of Marbel University 1st Semester SY 2025-2026
City of Koronadal Mrs. Brenda M. Balala, MIT

MIT 261– Parallel and Distributed Systems

Project Title

Parallel and Distributed Student Performance Analytics Platform with MongoDB

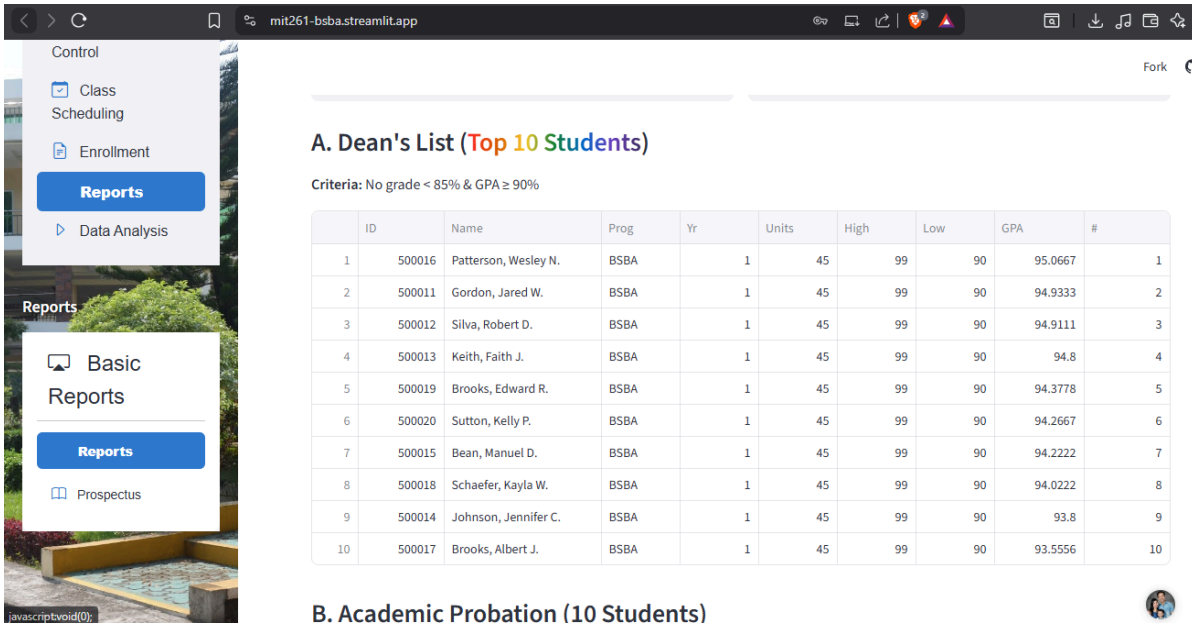
Group members: Assigned Tasked

1. **ALDEN QUIÑONES** **REGISTRAR**
2. **JUSTINE JOHN JAY DE ALA** **FACULTY**
3. **APRIL ANN BUDIONGAN** **STUDENT**

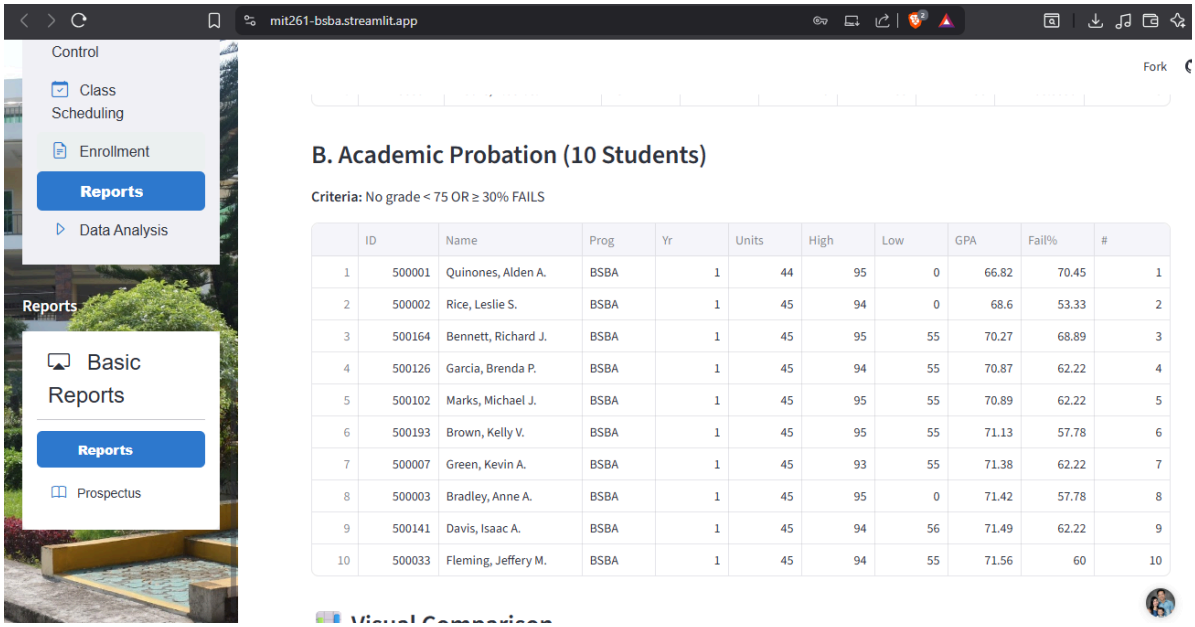
Contents

Registrar's Office Dashboard Reports

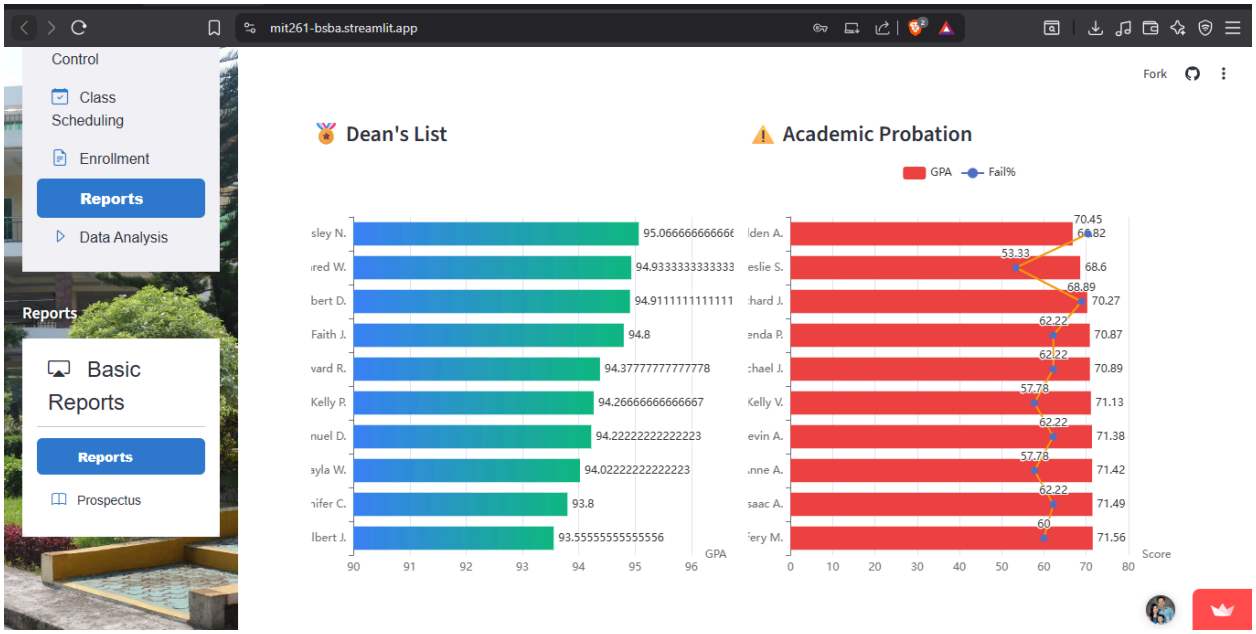
1. Student Academic Standing Report



	ID	Name	Prog	Yr	Units	High	Low	GPA	#
1	500016	Patterson, Wesley N.	BSBA	1	45	99	90	95.0667	1
2	500011	Gordon, Jared W.	BSBA	1	45	99	90	94.9333	2
3	500012	Silva, Robert D.	BSBA	1	45	99	90	94.9111	3
4	500013	Keith, Faith J.	BSBA	1	45	99	90	94.8	4
5	500019	Brooks, Edward R.	BSBA	1	45	99	90	94.3778	5
6	500020	Sutton, Kelly P.	BSBA	1	45	99	90	94.2667	6
7	500015	Bean, Manuel D.	BSBA	1	45	99	90	94.2222	7
8	500018	Schaefer, Kayla W.	BSBA	1	45	99	90	94.0222	8
9	500014	Johnson, Jennifer C.	BSBA	1	45	99	90	93.8	9
10	500017	Brooks, Albert J.	BSBA	1	45	99	90	93.5556	10

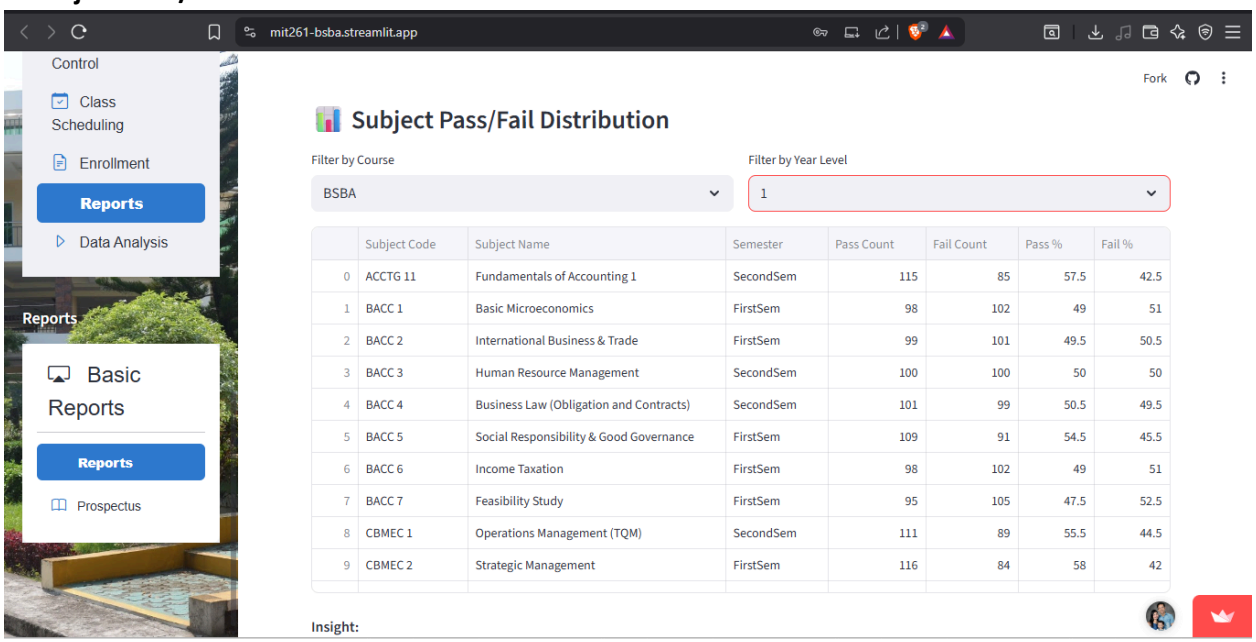


	ID	Name	Prog	Yr	Units	High	Low	GPA	Fail%	#
1	500001	Quinones, Alden A.	BSBA	1	44	95	0	66.82	70.45	1
2	500002	Rice, Leslie S.	BSBA	1	45	94	0	68.6	53.33	2
3	500164	Bennett, Richard J.	BSBA	1	45	95	55	70.27	68.89	3
4	500126	Garcia, Brenda P.	BSBA	1	45	94	55	70.87	62.22	4
5	500102	Marks, Michael J.	BSBA	1	45	95	55	70.89	62.22	5
6	500193	Brown, Kelly V.	BSBA	1	45	95	55	71.13	57.78	6
7	500007	Green, Kevin A.	BSBA	1	45	93	55	71.38	62.22	7
8	500003	Bradley, Anne A.	BSBA	1	45	95	0	71.42	57.78	8
9	500141	Davis, Isaac A.	BSBA	1	45	94	56	71.49	62.22	9
10	500033	Fleming, Jeffery M.	BSBA	1	45	94	55	71.56	60	10

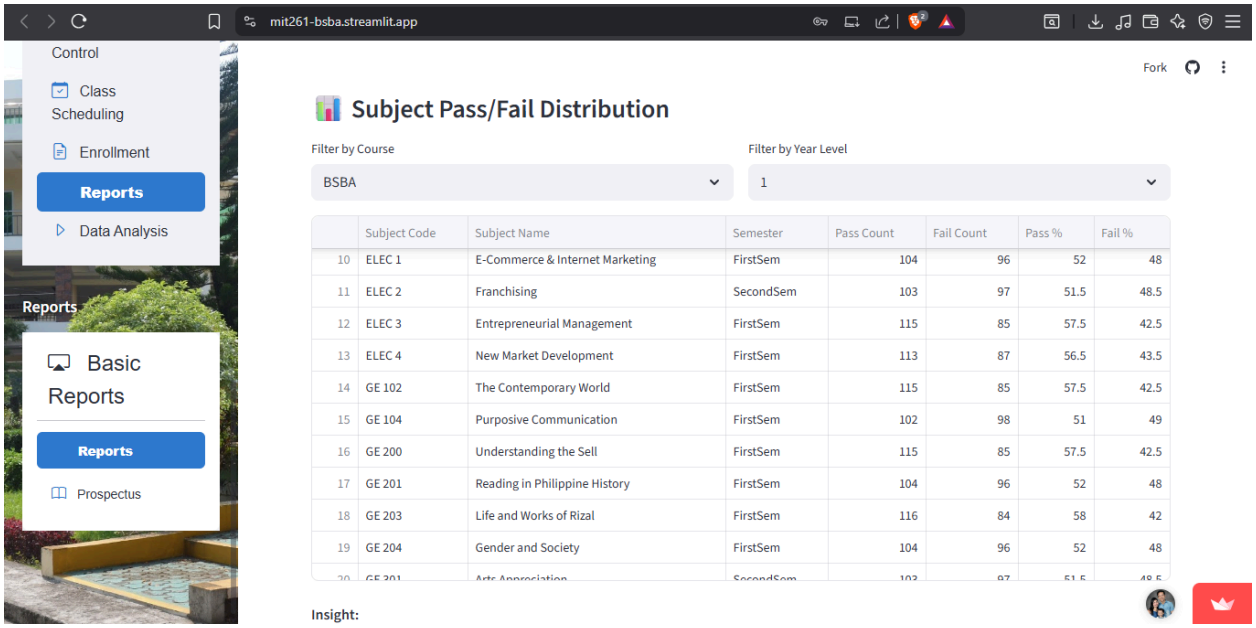


Insight: Enables quick identification of high achievers, students needing support, and overall academic distribution.

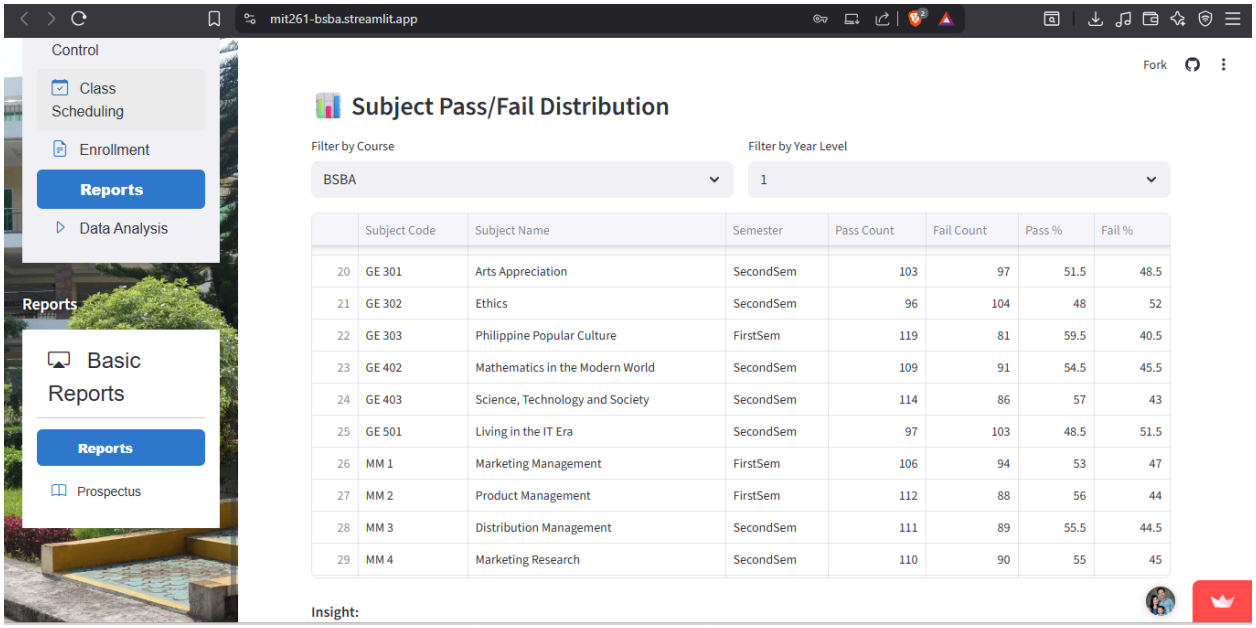
2. Subject Pass/Fail Distribution



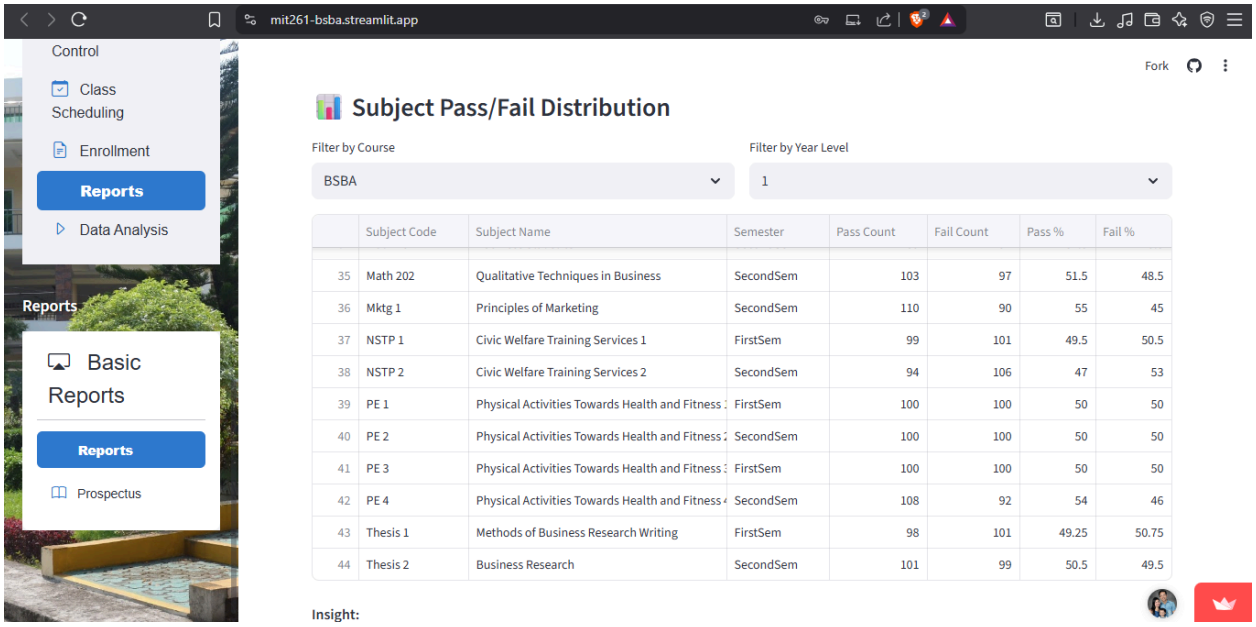
	Subject Code	Subject Name	Semester	Pass Count	Fail Count	Pass %	Fail %
0	ACCTG 11	Fundamentals of Accounting 1	SecondSem	115	85	57.5	42.5
1	BACC 1	Basic Microeconomics	FirstSem	98	102	49	51
2	BACC 2	International Business & Trade	FirstSem	99	101	49.5	50.5
3	BACC 3	Human Resource Management	SecondSem	100	100	50	50
4	BACC 4	Business Law (Obligation and Contracts)	SecondSem	101	99	50.5	49.5
5	BACC 5	Social Responsibility & Good Governance	FirstSem	109	91	54.5	45.5
6	BACC 6	Income Taxation	FirstSem	98	102	49	51
7	BACC 7	Feasibility Study	FirstSem	95	105	47.5	52.5
8	CBMEC 1	Operations Management (TQM)	SecondSem	111	89	55.5	44.5
9	CBMEC 2	Strategic Management	FirstSem	116	84	58	42

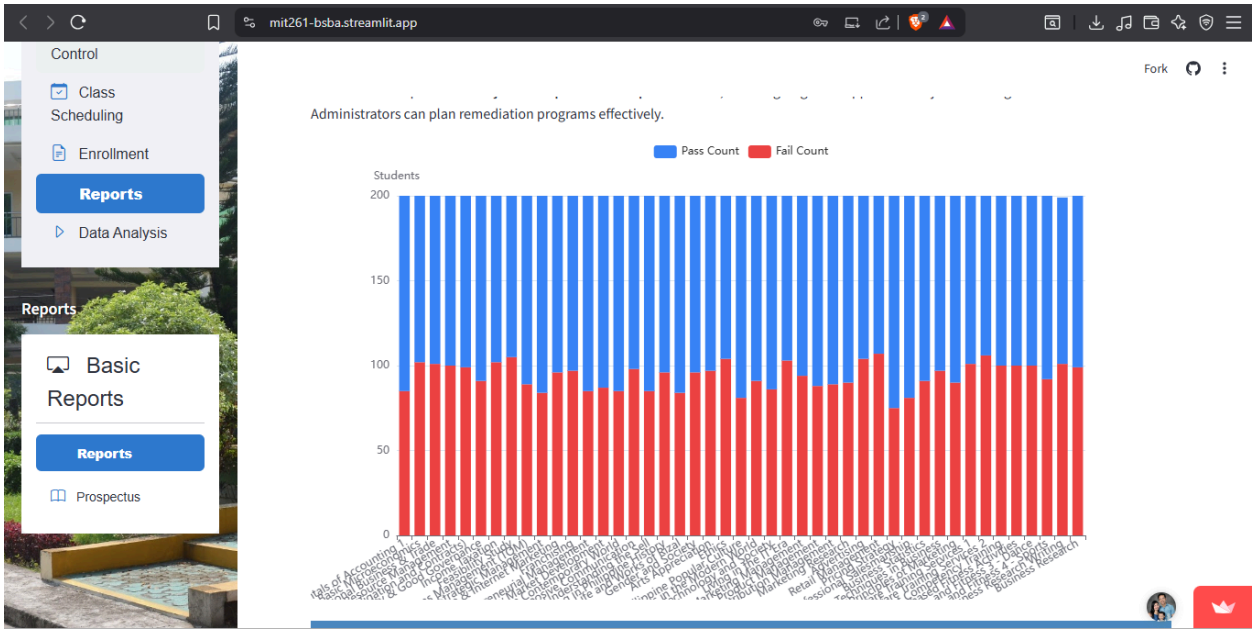


	Subject Code	Subject Name	Semester	Pass Count	Fail Count	Pass %	Fail %
10	ELEC 1	E-Commerce & Internet Marketing	FirstSem	104	96	52	48
11	ELEC 2	Franchising	SecondSem	103	97	51.5	48.5
12	ELEC 3	Entrepreneurial Management	FirstSem	115	85	57.5	42.5
13	ELEC 4	New Market Development	FirstSem	113	87	56.5	43.5
14	GE 102	The Contemporary World	FirstSem	115	85	57.5	42.5
15	GE 104	Purposive Communication	FirstSem	102	98	51	49
16	GE 200	Understanding the Sell	FirstSem	115	85	57.5	42.5
17	GE 201	Reading in Philippine History	FirstSem	104	96	52	48
18	GE 203	Life and Works of Rizal	FirstSem	116	84	58	42
19	GE 204	Gender and Society	FirstSem	104	96	52	48



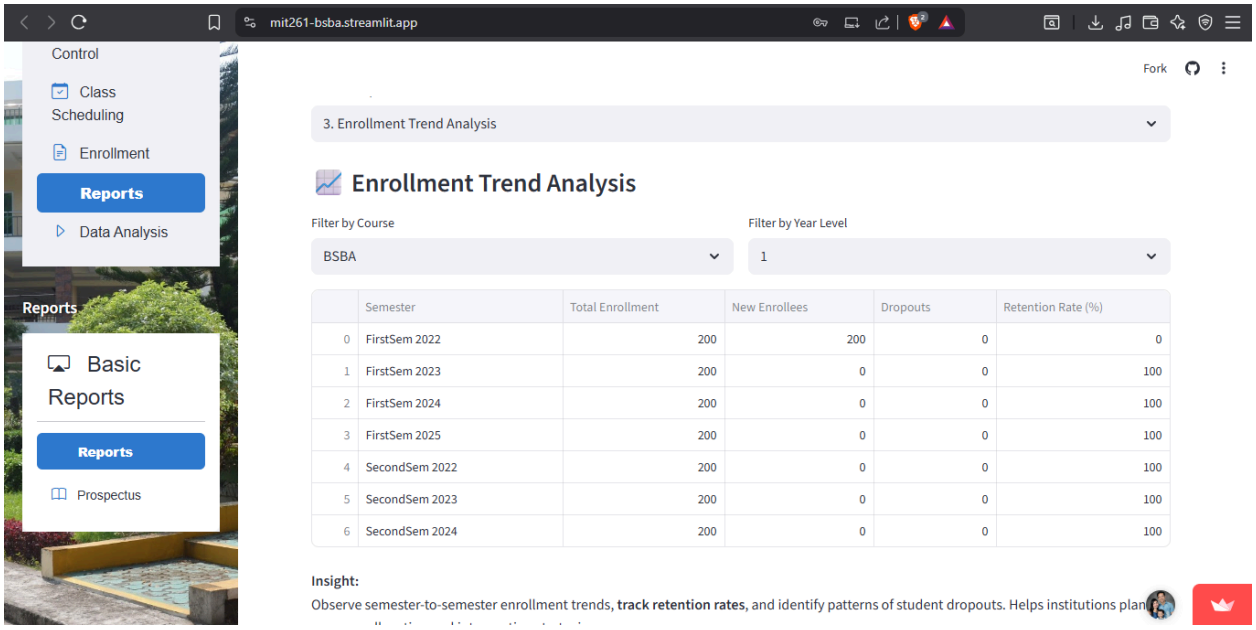
	Subject Code	Subject Name	Semester	Pass Count	Fail Count	Pass %	Fail %
20	GE 301	Arts Appreciation	SecondSem	103	97	51.5	48.5
21	GE 302	Ethics	SecondSem	96	104	48	52
22	GE 303	Philippine Popular Culture	FirstSem	119	81	59.5	40.5
23	GE 402	Mathematics in the Modern World	SecondSem	109	91	54.5	45.5
24	GE 403	Science, Technology and Society	SecondSem	114	86	57	43
25	GE 501	Living in the IT Era	SecondSem	97	103	48.5	51.5
26	MM 1	Marketing Management	FirstSem	106	94	53	47
27	MM 2	Product Management	FirstSem	112	88	56	44
28	MM 3	Distribution Management	SecondSem	111	89	55.5	44.5
29	MM 4	Marketing Research	SecondSem	110	90	55	45

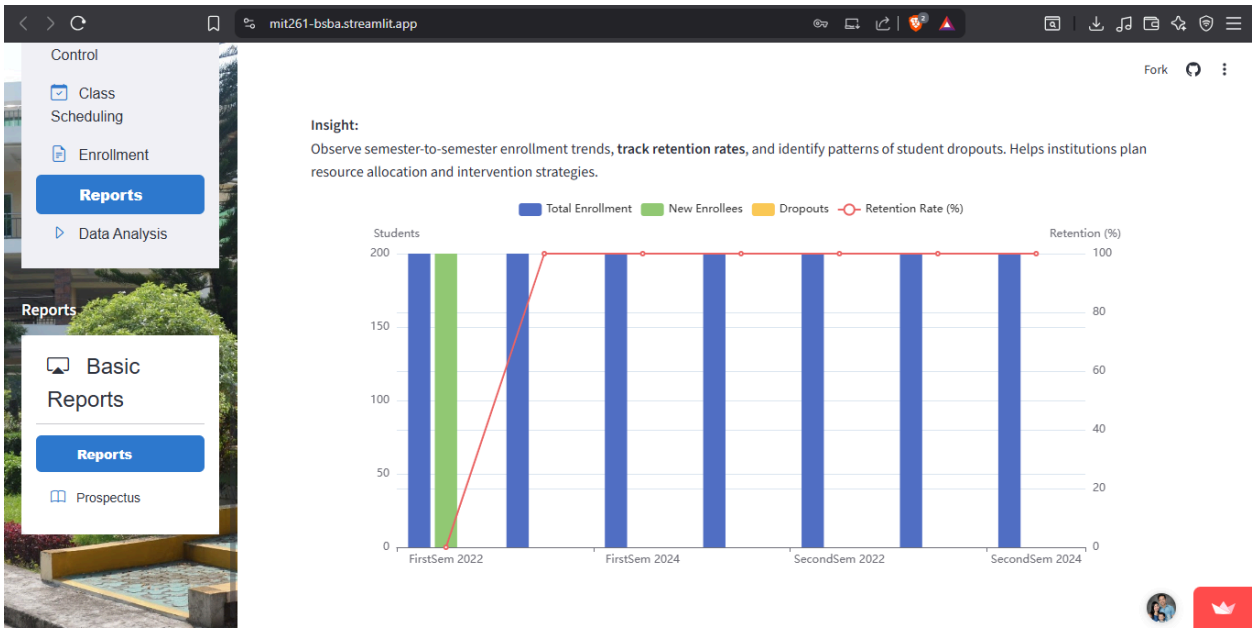




Insight: Clear visibility into subject-level performance by term—supports targeted interventions.

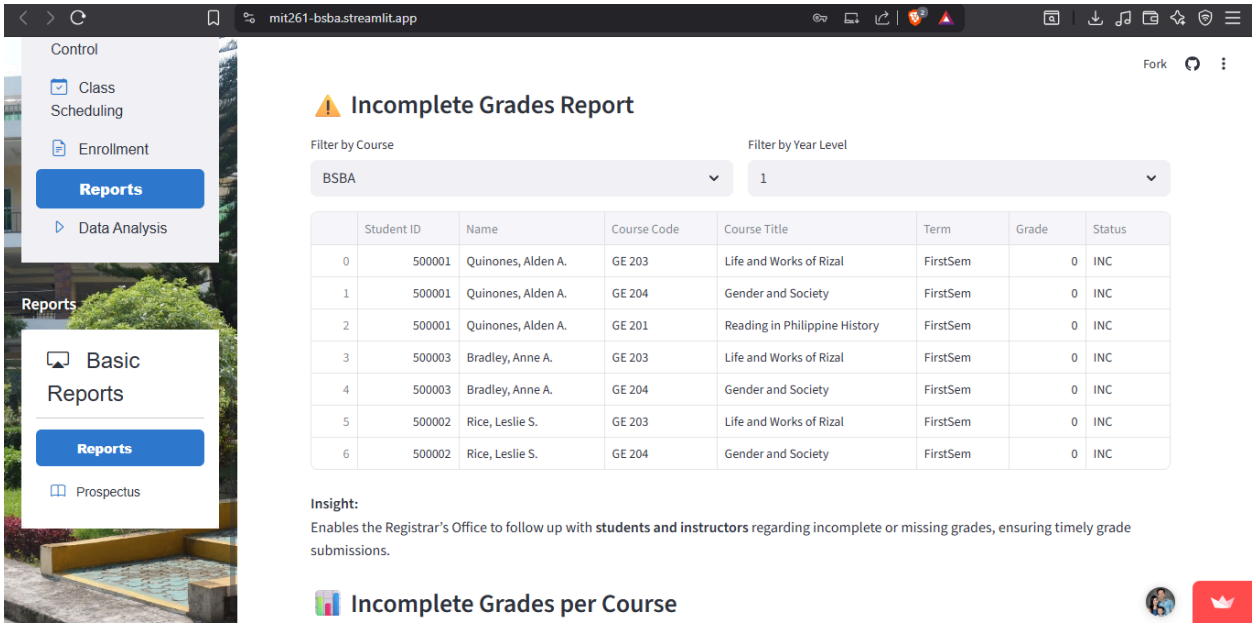
3. Enrollment Trend Analysis

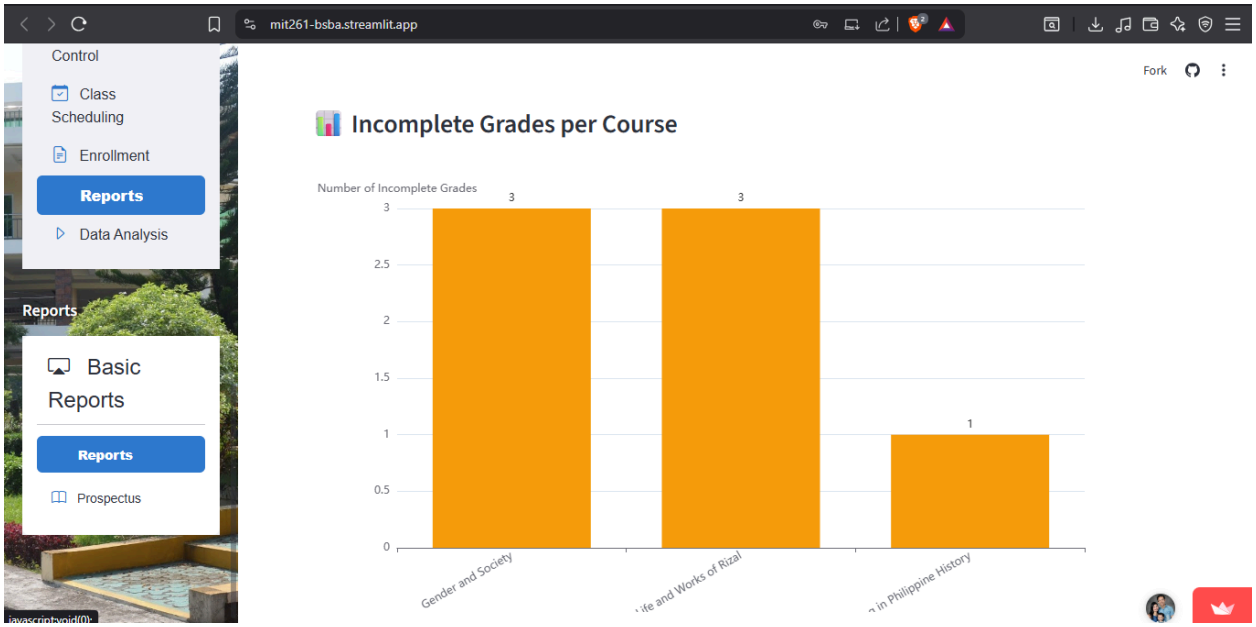




Insight: Plotting these values as line or area charts reveals enrollment dynamics—semester over semester or year over year.

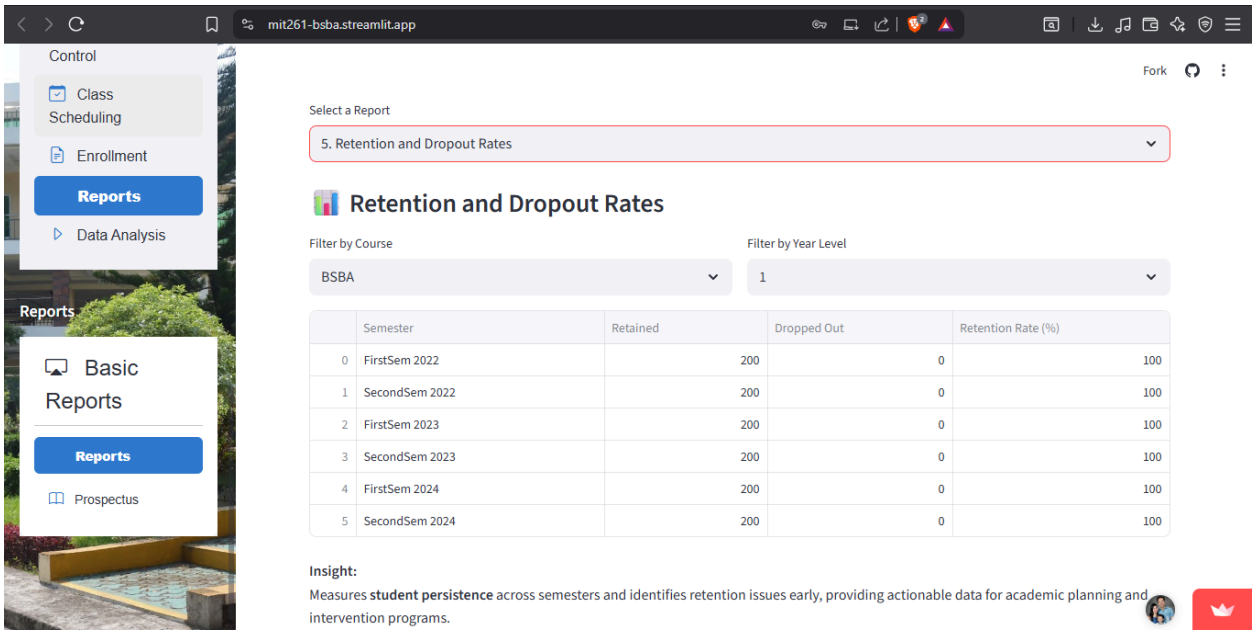
4. Incomplete Grades Report

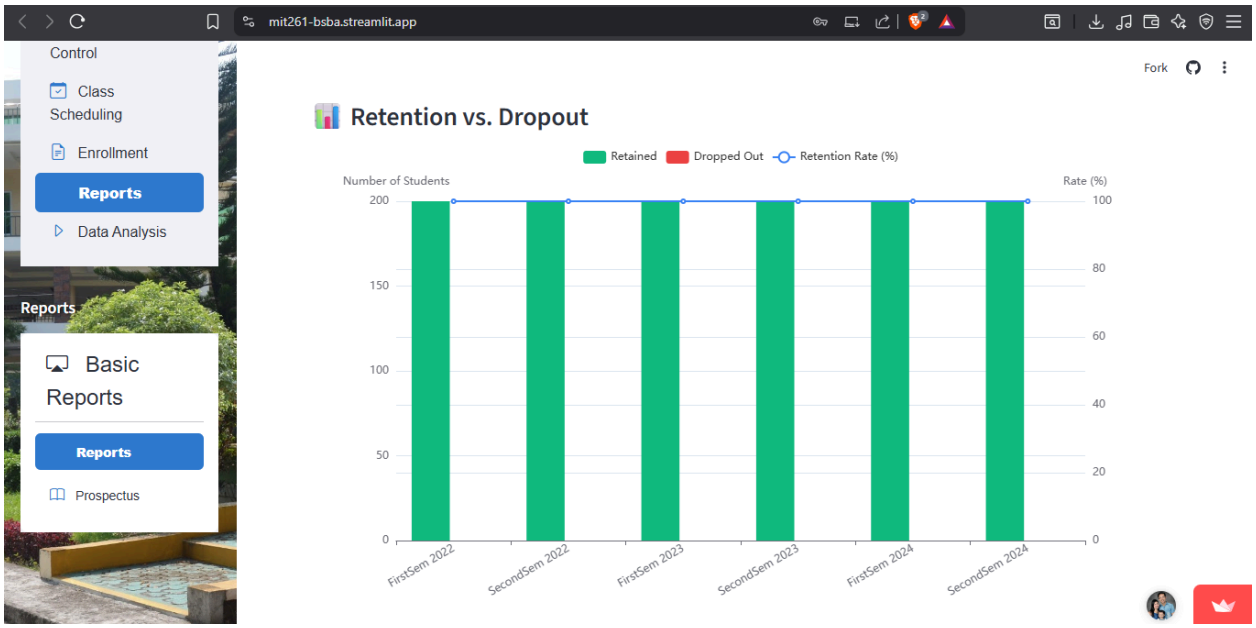




Insight: Helps Registrar's Office identify and follow up with students and instructors regarding incomplete or missing grade submissions.

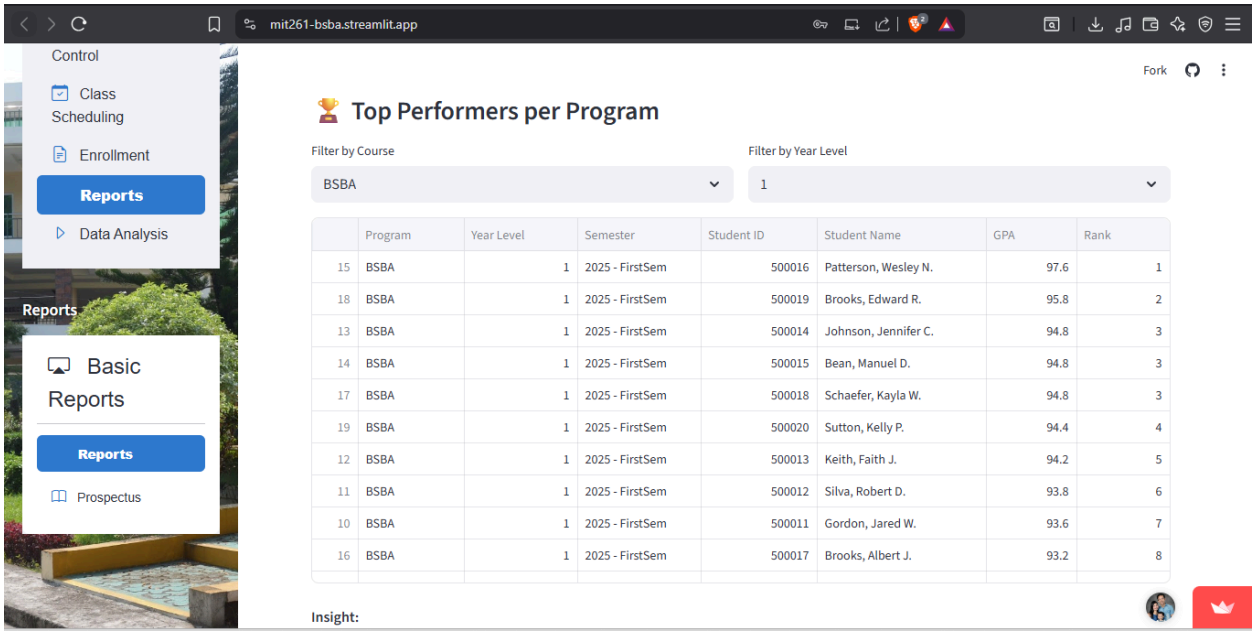
5. Retention and Dropout Rates

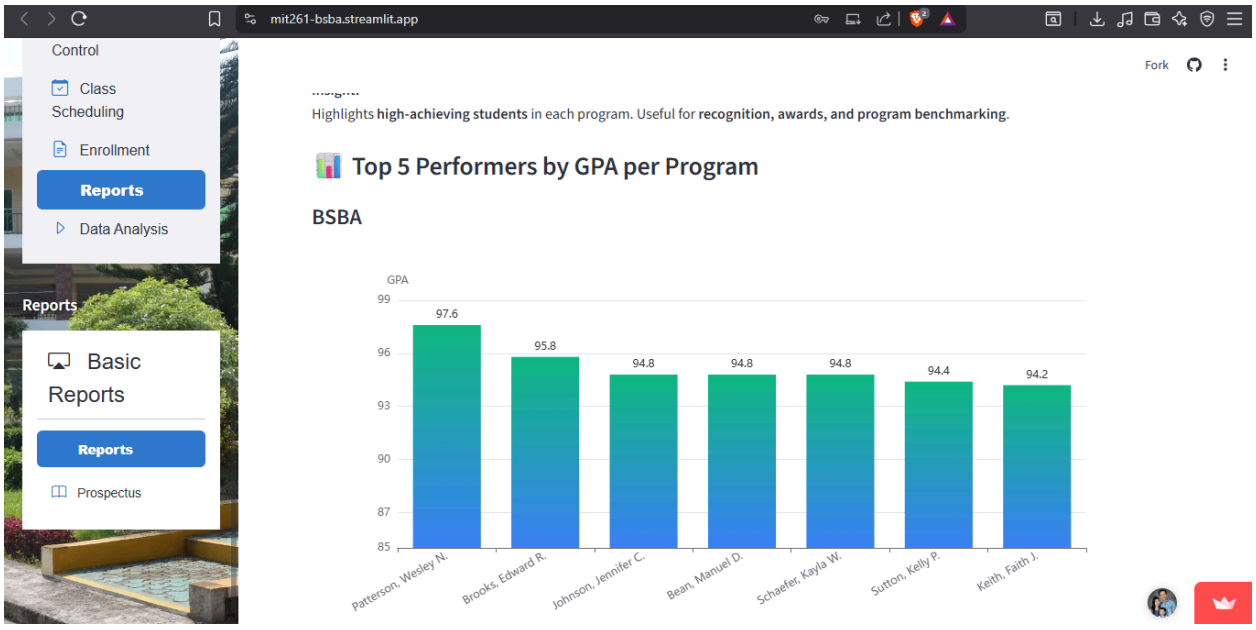




Insight: Measures student persistence and identifies retention issues early—critical for institutional planning.

6. Top Performers per Program

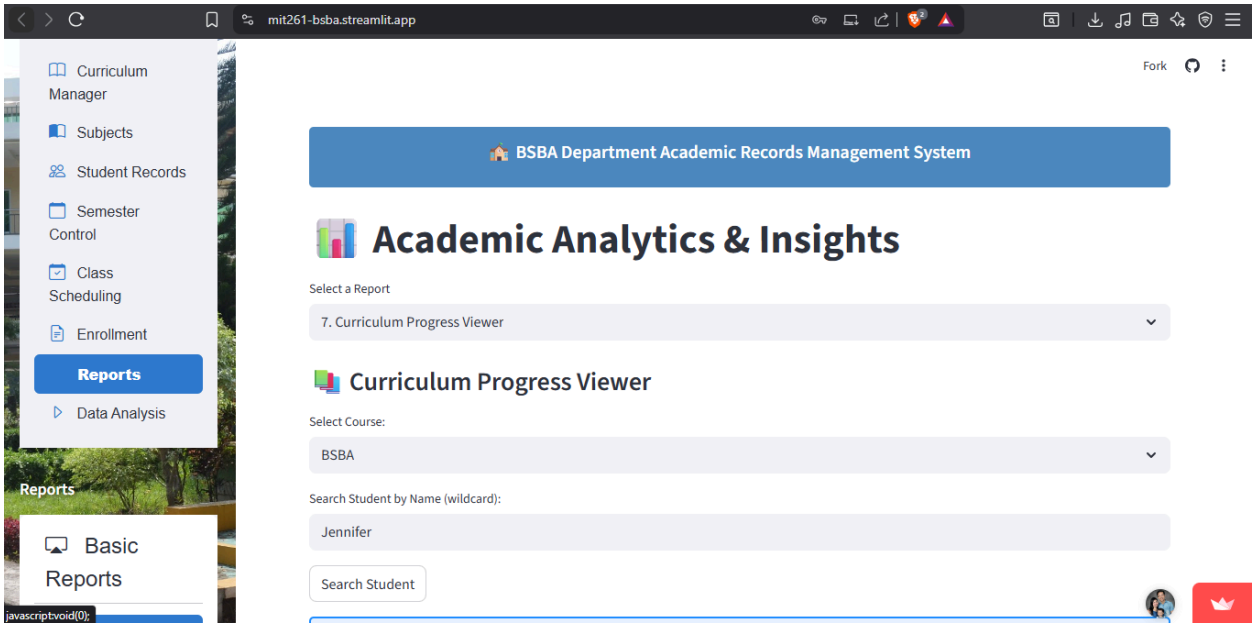


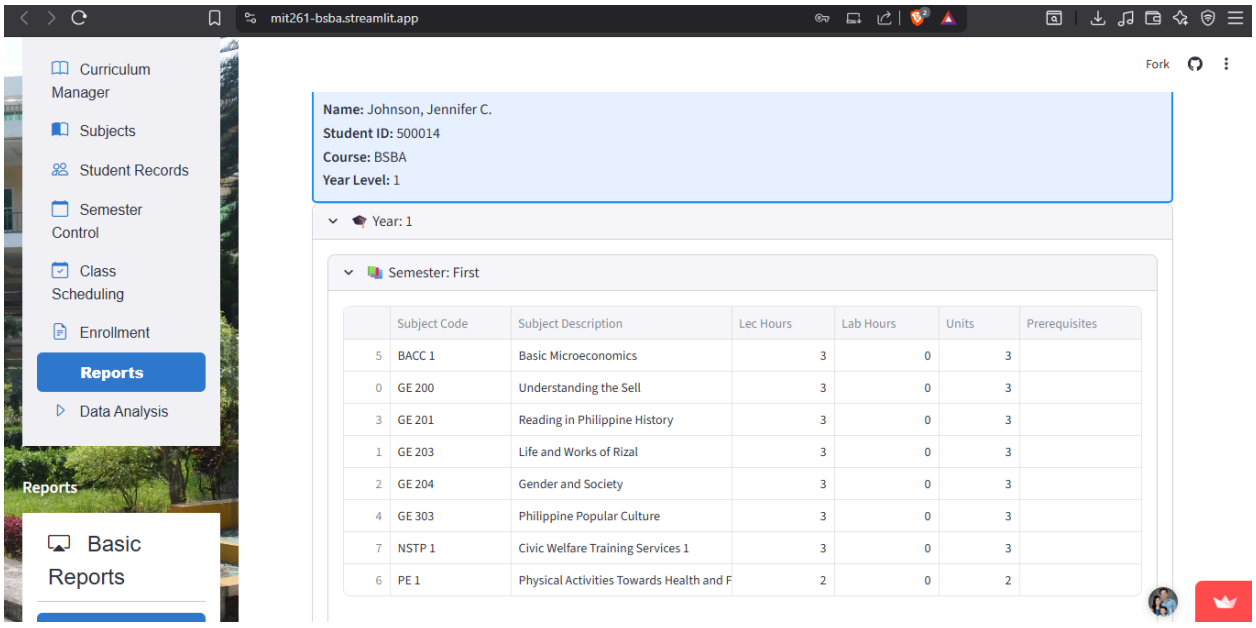


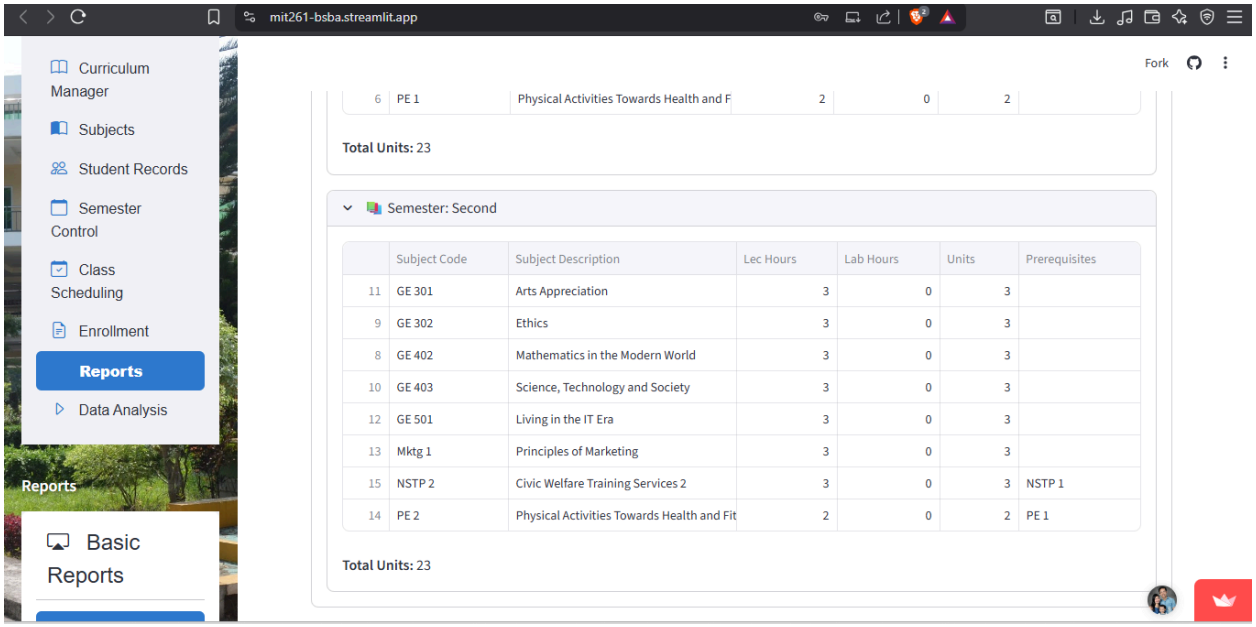
Insight: Highlights top academic performers by discipline—great for awards, recognitions, and program-level analysis.

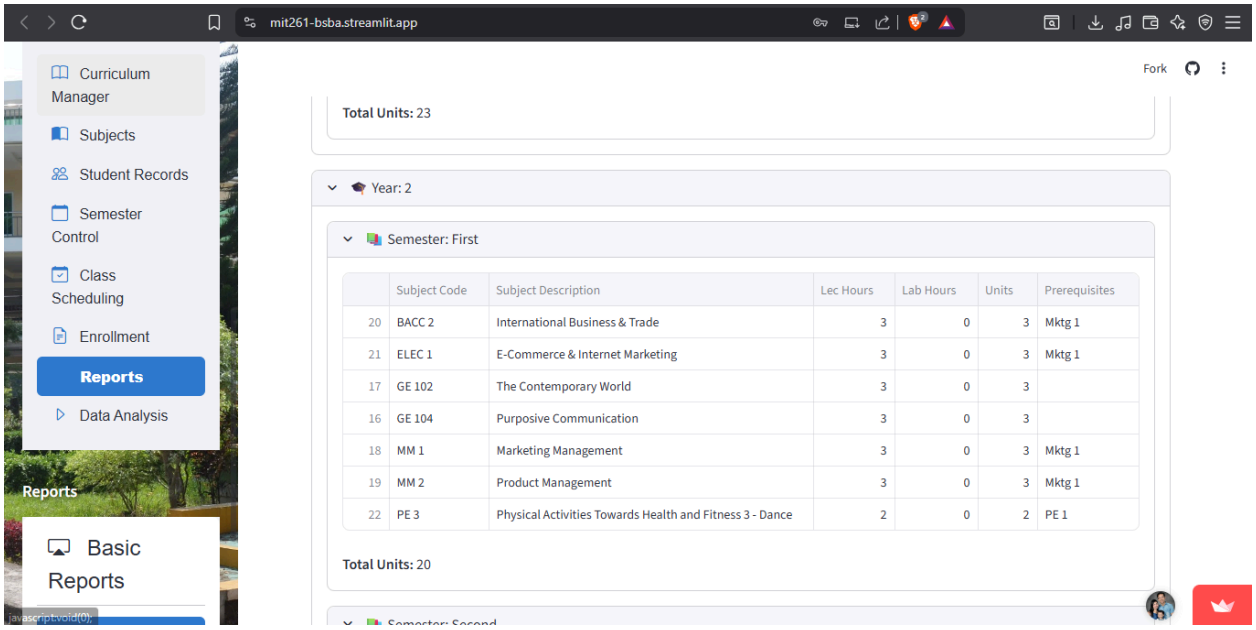
7. Curriculum Progress and Advising

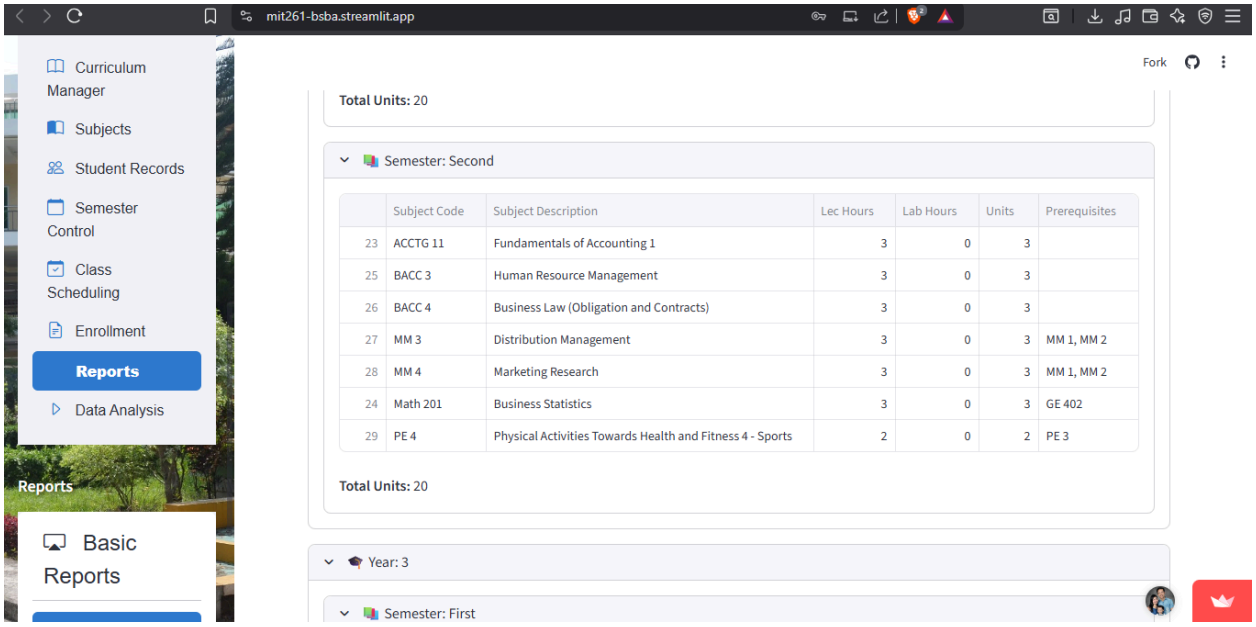
Filter by course and student name: BSBA- Jennifer C. Johnson

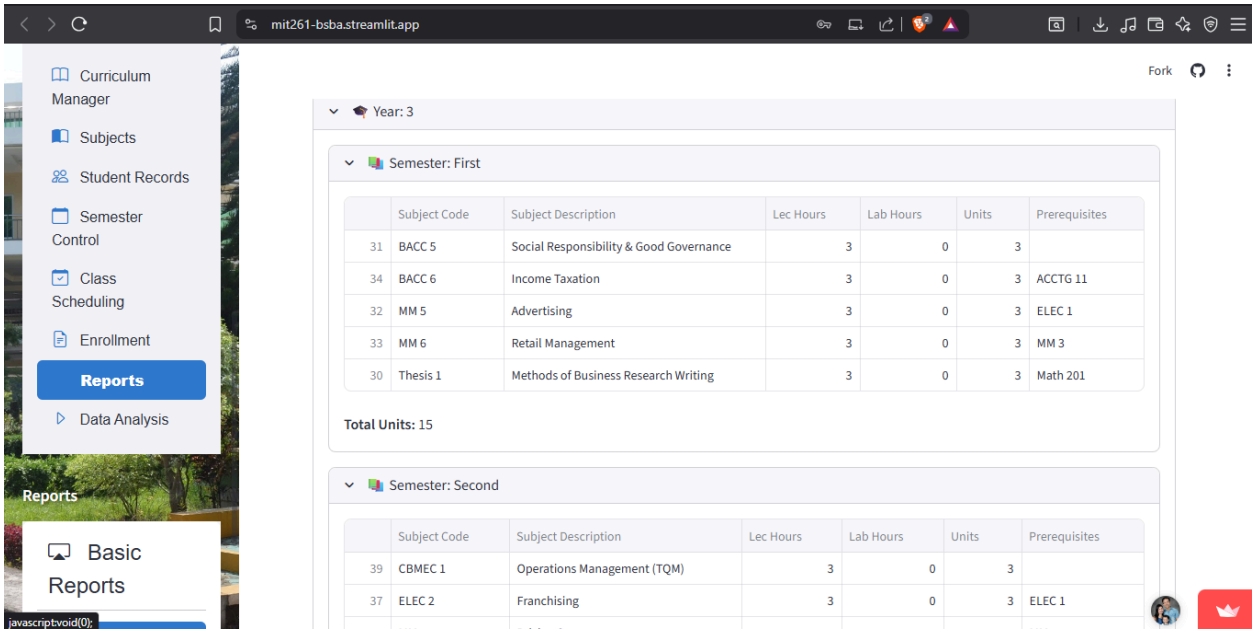


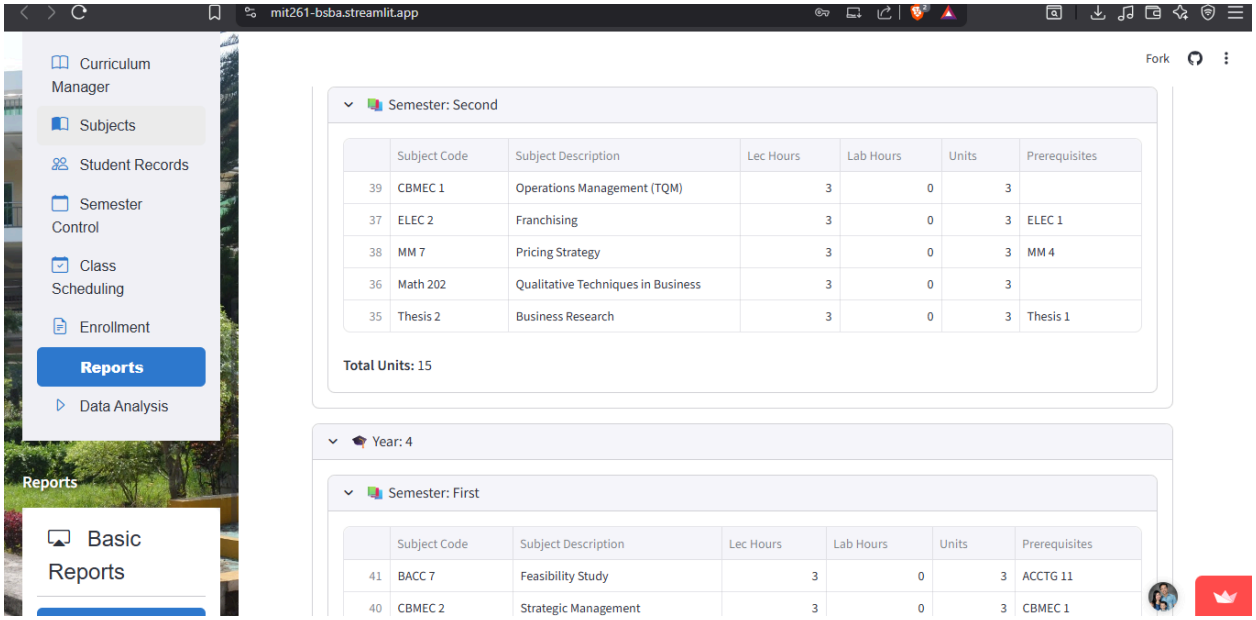


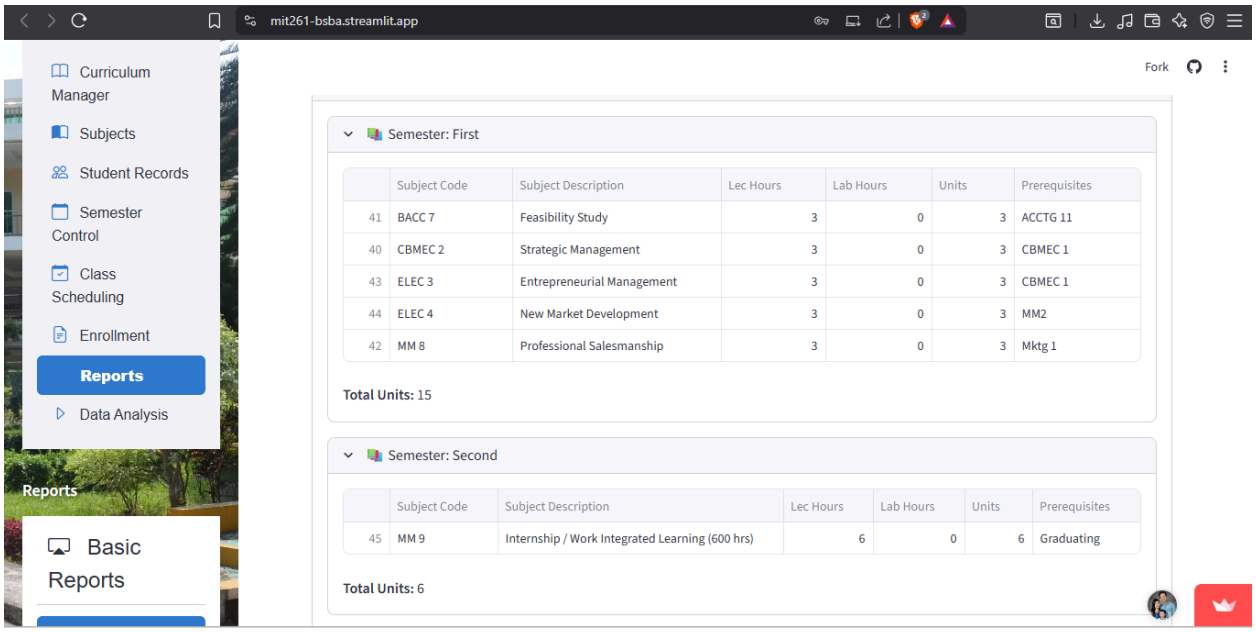

mit261-bsba.streamlit.app
Fork
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
6 PE 1 Physical Activities Towards Health and F 2 0 2
Total Units: 23
Semester: Second
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
11 GE 301 Arts Appreciation 3 0 3
9 GE 302 Ethics 3 0 3
8 GE 402 Mathematics in the Modern World 3 0 3
10 GE 403 Science, Technology and Society 3 0 3
12 GE 501 Living in the IT Era 3 0 3
13 Mktg 1 Principles of Marketing 3 0 3
15 NSTP 2 Civic Welfare Training Services 2 3 0 3 NSTP 1
14 PE 2 Physical Activities Towards Health and Fit 2 0 2 PE 1
Total Units: 23

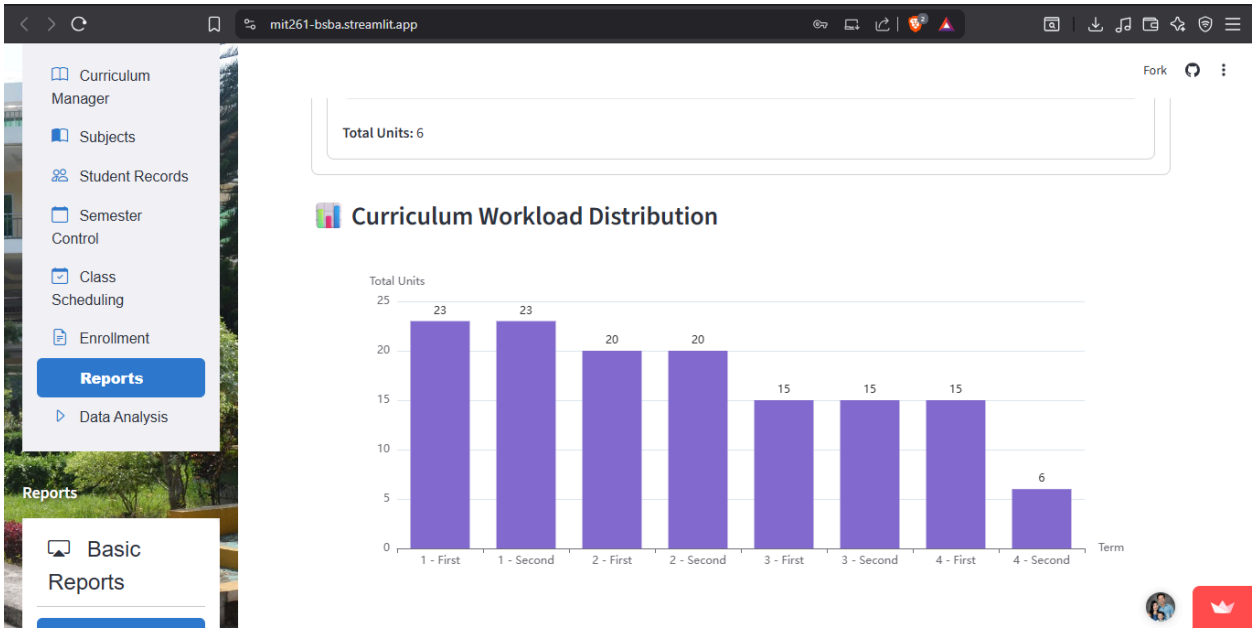

mit261-bsba.streamlit.app
Fork
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
Total Units: 23
Year: 2
Semester: First
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
20 BACC 2 International Business & Trade 3 0 3 Mktg 1
21 ELEC 1 E-Commerce & Internet Marketing 3 0 3 Mktg 1
17 GE 102 The Contemporary World 3 0 3
16 GE 104 Purposive Communication 3 0 3
18 MM 1 Marketing Management 3 0 3 Mktg 1
19 MM 2 Product Management 3 0 3 Mktg 1
22 PE 3 Physical Activities Towards Health and Fitness 3 - Dance 2 0 2 PE 1
Total Units: 20


mit261-bsba.streamlit.app
Fork
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
Total Units: 20
Semester: Second
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
23 | ACCTG 11 | Fundamentals of Accounting 1 | 3 | 0 | 3 |
25 | BACC 3 | Human Resource Management | 3 | 0 | 3 |
26 | BACC 4 | Business Law (Obligation and Contracts) | 3 | 0 | 3 |
27 | MM 3 | Distribution Management | 3 | 0 | 3 | MM 1, MM 2
28 | MM 4 | Marketing Research | 3 | 0 | 3 | MM 1, MM 2
24 | Math 201 | Business Statistics | 3 | 0 | 3 | GE 402
29 | PE 4 | Physical Activities Towards Health and Fitness 4 - Sports | 2 | 0 | 2 | PE 3
Total Units: 20
Year: 3
Semester: First


mit261-bsba.streamlit.app
Fork
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
javascript:void(0);
Year: 3
Semester: First
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
31 | BACC 5 | Social Responsibility & Good Governance | 3 | 0 | 3 |
34 | BACC 6 | Income Taxation | 3 | 0 | 3 | ACCTG 11
32 | MM 5 | Advertising | 3 | 0 | 3 | ELEC 1
33 | MM 6 | Retail Management | 3 | 0 | 3 | MM 3
30 | Thesis 1 | Methods of Business Research Writing | 3 | 0 | 3 | Math 201
Total Units: 15
Semester: Second
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
39 | CBMEC 1 | Operations Management (TQM) | 3 | 0 | 3 |
37 | ELEC 2 | Franchising | 3 | 0 | 3 | ELEC 1


mit261-bsba.streamlit.app
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
Fork
Semester: Second
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
39 | CBMEC 1 | Operations Management (TQM) | 3 | 0 | 3 |
37 | ELEC 2 | Franchising | 3 | 0 | 3 | ELEC 1
38 | MM 7 | Pricing Strategy | 3 | 0 | 3 | MM 4
36 | Math 202 | Qualitative Techniques in Business | 3 | 0 | 3 |
35 | Thesis 2 | Business Research | 3 | 0 | 3 | Thesis 1
Total Units: 15
Year: 4
Semester: First
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
41 | BACC 7 | Feasibility Study | 3 | 0 | 3 | ACCTG 11
40 | CBMEC 2 | Strategic Management | 3 | 0 | 3 | CBMEC 1

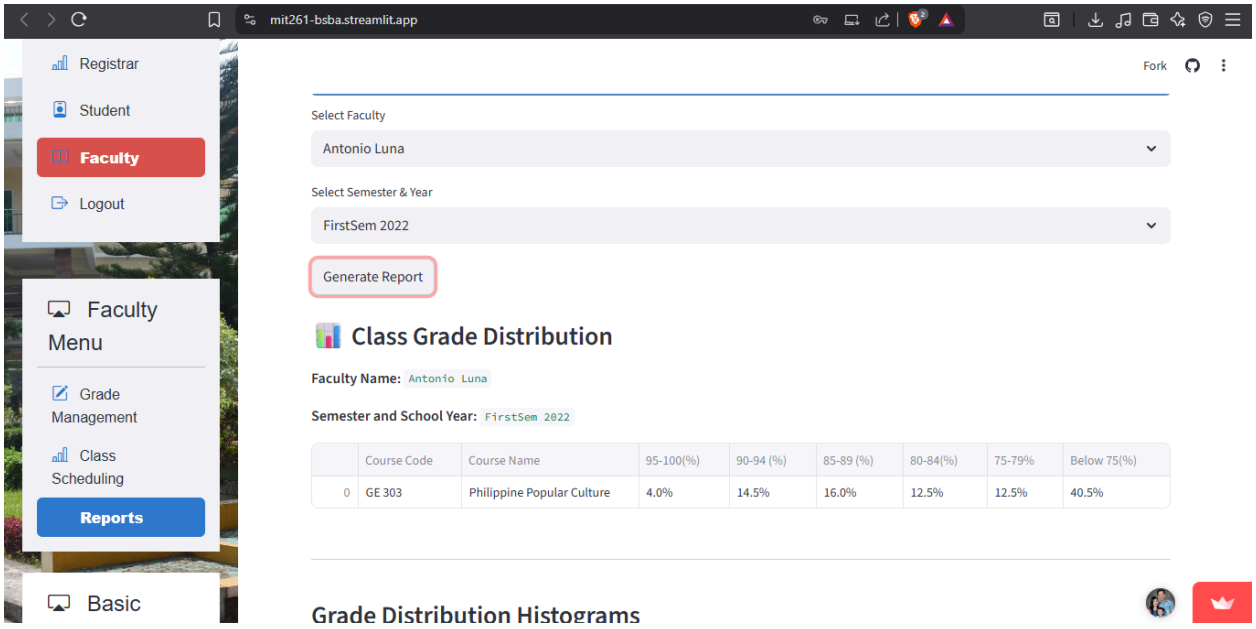

mit261-bsba.streamlit.app
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
Fork
Semester: First
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
41 | BACC 7 | Feasibility Study | 3 | 0 | 3 | ACCTG 11
40 | CBMEC 2 | Strategic Management | 3 | 0 | 3 | CBMEC 1
43 | ELEC 3 | Entrepreneurial Management | 3 | 0 | 3 | CBMEC 1
44 | ELEC 4 | New Market Development | 3 | 0 | 3 | MM2
42 | MM 8 | Professional Salesmanship | 3 | 0 | 3 | Mktg 1
Total Units: 15
Semester: Second
Subject Code | Subject Description | Lec Hours | Lab Hours | Units | Prerequisites
45 | MM 9 | Internship / Work Integrated Learning (600 hrs) | 6 | 0 | 6 | Graduating
Total Units: 6

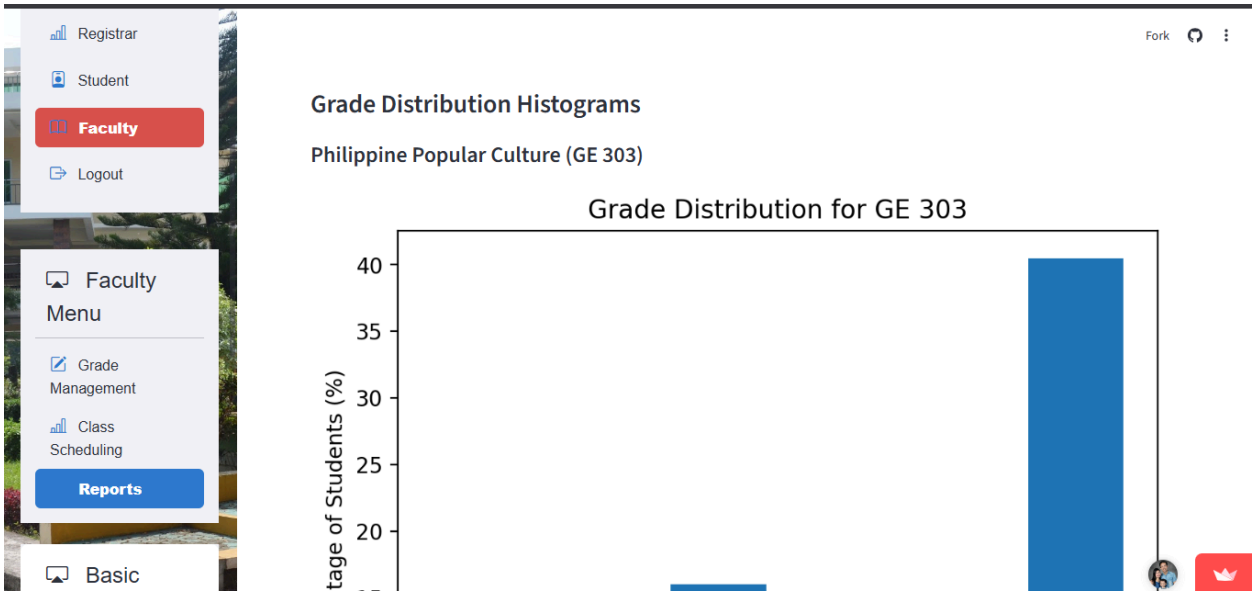

mit261-bsba.streamlit.app
Curriculum Manager
Subjects
Student Records
Semester Control
Class Scheduling
Enrollment
Reports
Data Analysis
Reports
Basic Reports
Fork
Total Units: 6
Curriculum Workload Distribution
Total Units
25
20
15
10
5
0
23
23
20
20
15
15
15
6
1 - First
1 - Second
2 - First
2 - Second
3 - First
3 - Second
4 - First
4 - Second
Term

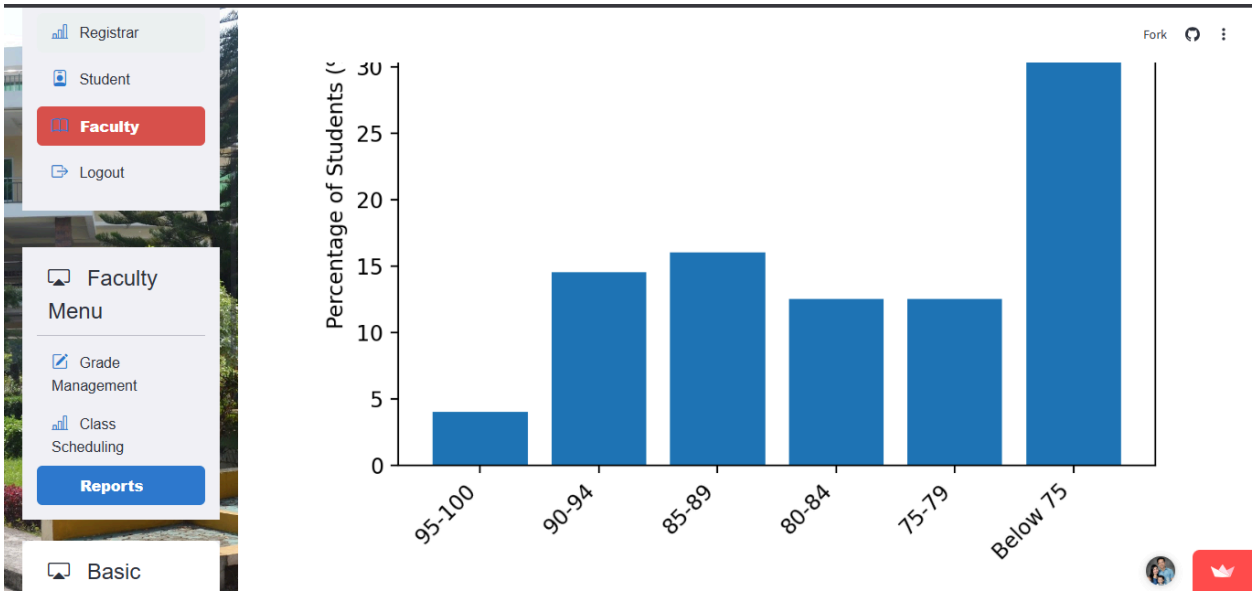
Faculty Dashboard Reports

1. **Class Grade Distribution**

- **Displays a histogram of student grade distribution per subject.**
  Faculty Name: <u>Antonio Luna</u>
  Semester and School Year: <u>FirstSem 2022</u>





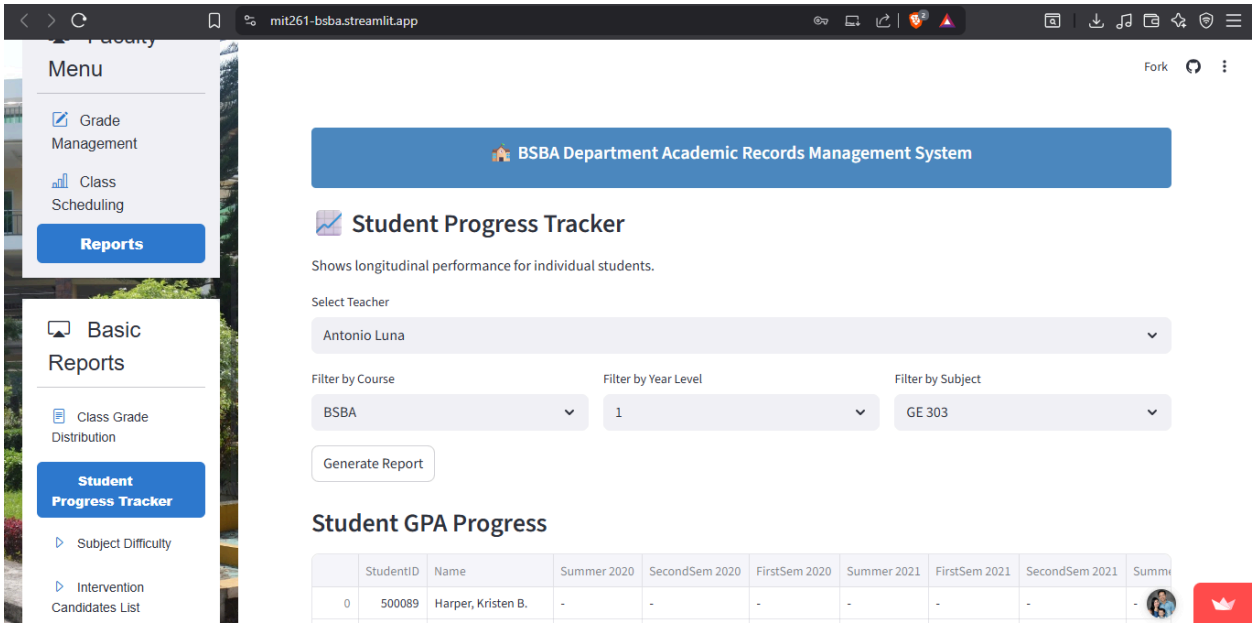


Insight: Quick view of overall performance—identifies courses with skewed grade distributions or those with high failure rates.

2. Student Progress Tracker

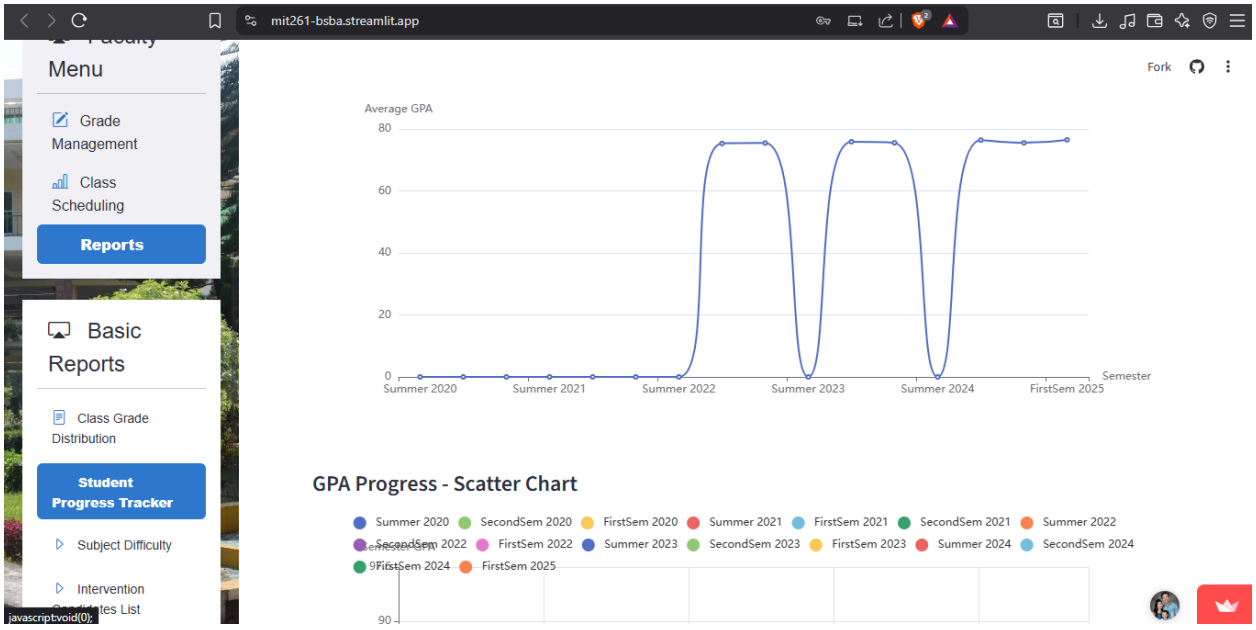
Shows longitudinal performance for individual students.

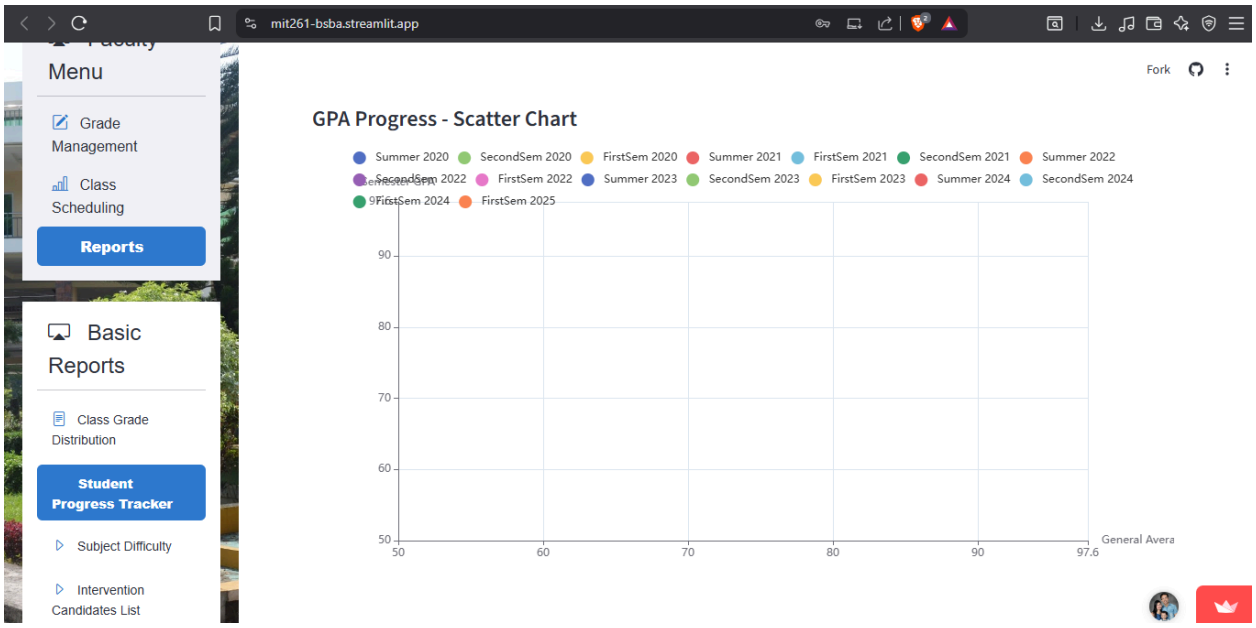
Filtered by Subject or course or YearLevel



	StudentID	Name	Summer 2020	SecondSem 2020	FirstSem 2020	Summer 2021	FirstSem 2021	SecondSem 2021	Summer 2022	SecondSem 2022	FirstSem 2022	Summer 2023	Secor
0	500089	Harper, Kristen B.	-	-	-	-	-	-	-	74.130435	75.739130	-	
1	500019	Brooks, Edward R.	-	-	-	-	-	-	-	93.173913	95.173913	-	
2	500068	Jacobson, Gregory V.	-	-	-	-	-	-	-	70.434783	78.000000	-	
3	500080	French, Andrew C.	-	-	-	-	-	-	-	72.913043	77.304348	-	
4	500099	Davis, Elizabeth D.	-	-	-	-	-	-	-	75.565217	69.608696	-	
5	500006	Benson, Gerald T.	-	-	-	-	-	-	-	73.565217	69.826087	-	
6	500007	Green, Kevin A.	-	-	-	-	-	-	-	68.000000	68.478261	-	
7	500023	Massey, Anthony W.	-	-	-	-	-	-	-	70.782609	76.260870	-	
8	500064	Marquez, Melissa B.	-	-	-	-	-	-	-	75.434783	74.521739	-	
9	500071	Zamora, Mackenzie K.	-	-	-	-	-	-	-	70.695652	78.347826	-	
10	500073	Wall, Elaine M.	-	-	-	-	-	-	-	76.521739	78.086957	-	
11	500082	Diaz, Randy A.	-	-	-	-	-	-	-	67.652174	80.565217	-	
12	500091	Chan, Jennifer L.	-	-	-	-	-	-	-	78.782609	83.130435	-	
13	500016	Patterson, Wesley N.	-	-	-	-	-	-	-	94.956522	93.000000	-	
14	500029	Barnett, Jamie J.	-	-	-	-	-	-	-	72.782609	80.695652	-	
15	500038	Walker, Sarah J.								67.086957	75.000000		

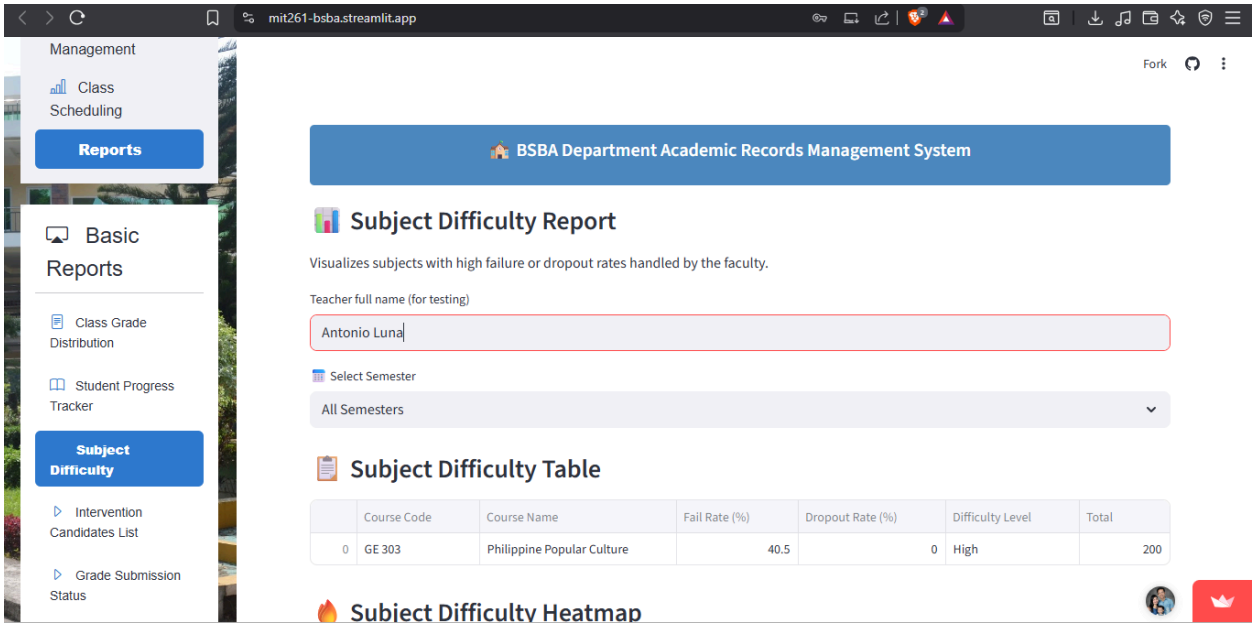
	2021	SecondSem 2021	Summer 2022	SecondSem 2022	FirstSem 2022	Summer 2023	SecondSem 2023	FirstSem 2023	Summer 2024	SecondSem 2024	FirstSem 2024	FirstSem 2025	Overall Trend
0		-	-	74.130435	75.739130	-	78.600000	76.950000	-	79.400000	73.000000	72.200000	Stable High
1		-	-	93.173913	95.173913	-	95.400000	93.800000	-	94.600000	93.000000	95.800000	Stable High
2		-	-	70.434783	78.000000	-	74.800000	75.200000	-	72.600000	86.600000	74.600000	Stable High
3		-	-	72.913043	77.304348	-	65.100000	76.800000	-	80.400000	70.400000	75.000000	Stable High
4		-	-	75.565217	69.608696	-	80.000000	72.550000	-	74.600000	67.800000	84.200000	Stable High
5		-	-	73.565217	69.826087	-	76.050000	76.750000	-	74.800000	66.400000	73.200000	Stable High
6		-	-	68.000000	68.478261	-	69.300000	74.300000	-	68.800000	73.600000	78.000000	Stable High
7		-	-	70.782609	76.260870	-	72.500000	74.350000	-	69.000000	75.400000	78.200000	Stable High
8		-	-	75.434783	74.521739	-	68.950000	81.150000	-	71.800000	69.400000	87.800000	Stable High
9		-	-	70.695652	78.347826	-	77.050000	73.450000	-	82.000000	76.600000	72.200000	Stable High
10		-	-	76.521739	78.086957	-	77.250000	75.050000	-	74.800000	80.800000	82.000000	Stable High
11		-	-	67.652174	80.565217	-	82.150000	72.250000	-	77.200000	70.200000	71.800000	Stable High
12		-	-	78.782609	83.130435	-	77.000000	77.550000	-	87.200000	81.800000	79.000000	Stable High
13		-	-	94.956522	93.000000	-	94.350000	94.350000	-	97.000000	96.000000	97.600000	Stable High
14		-	-	72.782609	80.695652	-	75.400000	75.250000	-	69.400000	74.600000	70.800000	Stable High
15				67.086957	75.000000		74.450000	78.350000		77.600000	80.800000	72.600000	

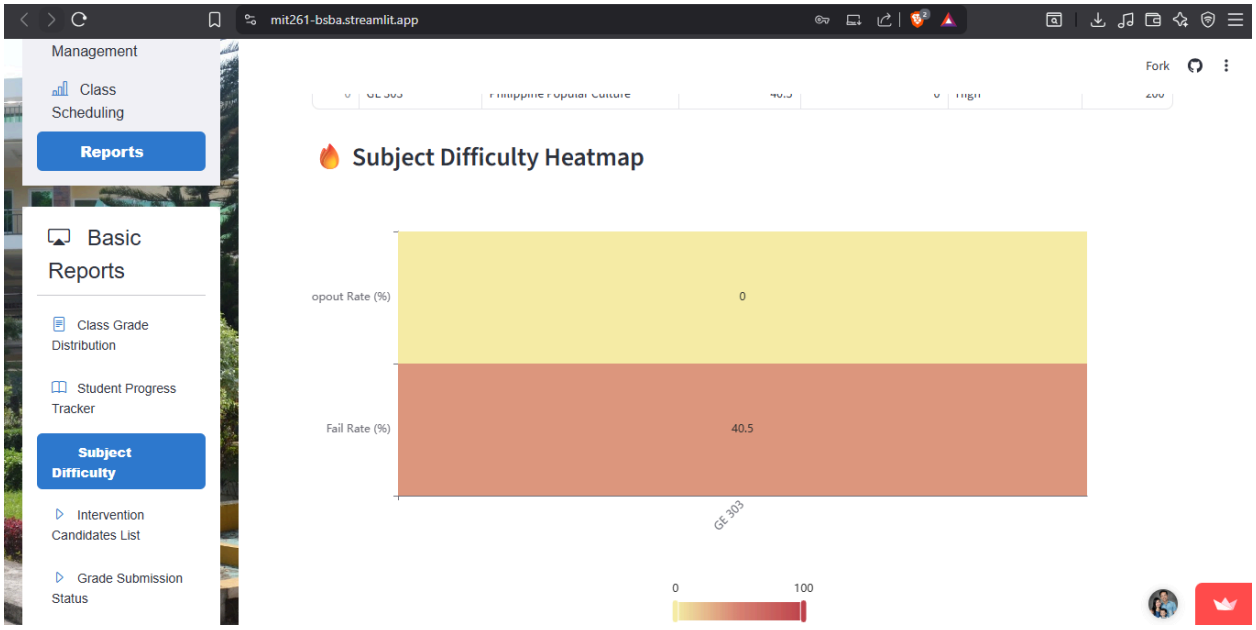

mit261-bsba.streamlit.app
Menu
Grade Management
Class Scheduling
Reports
Basic Reports
Class Grade Distribution
Student Progress Tracker
Subject Difficulty
Intervention
Fork
Average GPA
80
60
40
20
0
Summer 2020
Summer 2021
Summer 2022
Summer 2023
Summer 2024
FirstSem 2025
Semester
GPA Progress - Scatter Chart
Summer 2020
SecondSem 2020
FirstSem 2020
Summer 2021
FirstSem 2021
SecondSem 2021
Summer 2022
SecondSem 2022
FirstSem 2022
Summer 2023
SecondSem 2023
FirstSem 2023
Summer 2024
SecondSem 2024
FirstSem 2024
FirstSem 2025
90


mit261-bsba.streamlit.app
Menu
Grade Management
Class Scheduling
Reports
Basic Reports
Class Grade Distribution
Student Progress Tracker
Subject Difficulty
Intervention
Fork
GPA Progress - Scatter Chart
Summer 2020
SecondSem 2020
FirstSem 2020
Summer 2021
FirstSem 2021
SecondSem 2021
Summer 2022
SecondSem 2022
FirstSem 2022
Summer 2023
SecondSem 2023
FirstSem 2023
Summer 2024
SecondSem 2024
FirstSem 2024
FirstSem 2025
90
80
70
60
50
50
60
70
80
90
97.6
General Avera

3. **Subject Difficulty Heatmap**

- **Visualizes subjects with high failure or dropouts**
- **Subjects handled by the faculty**

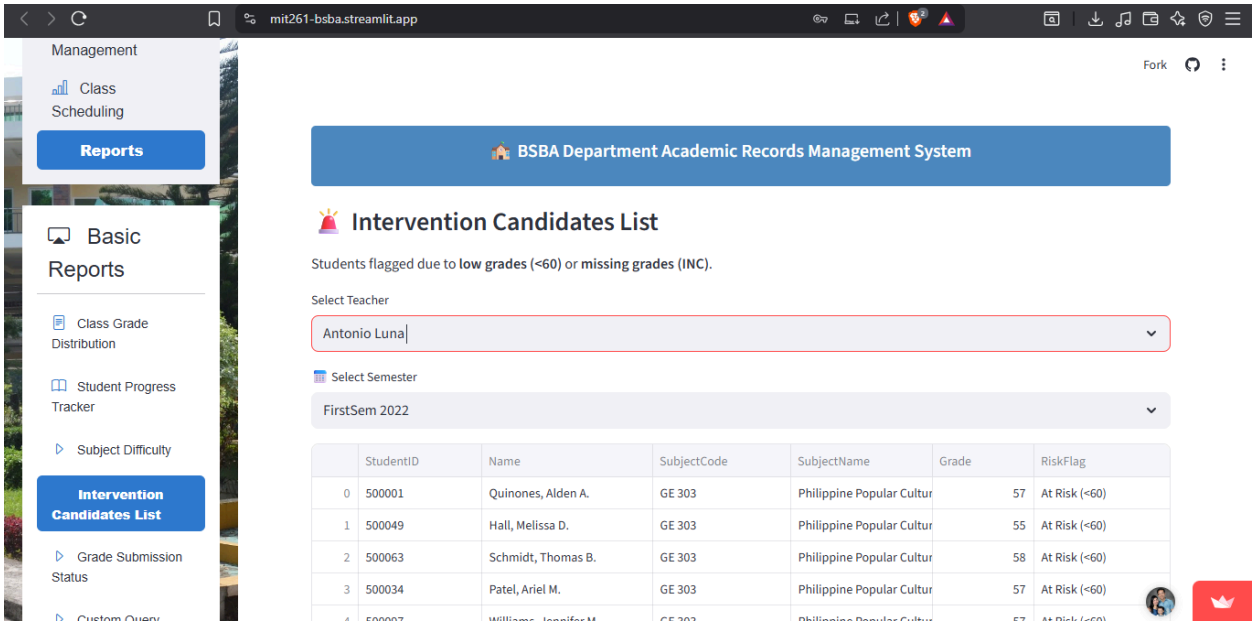


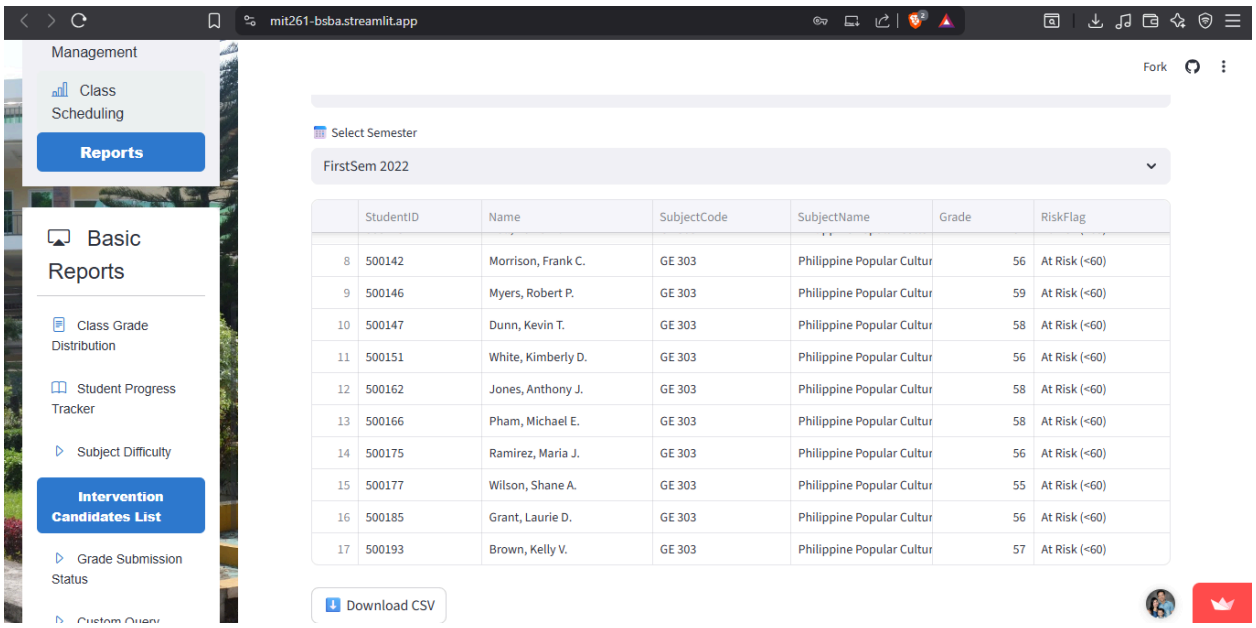


4. Intervention Candidates List

- **Lists students at academic risk based on current semester data (e.g. low grades, missing grades)**

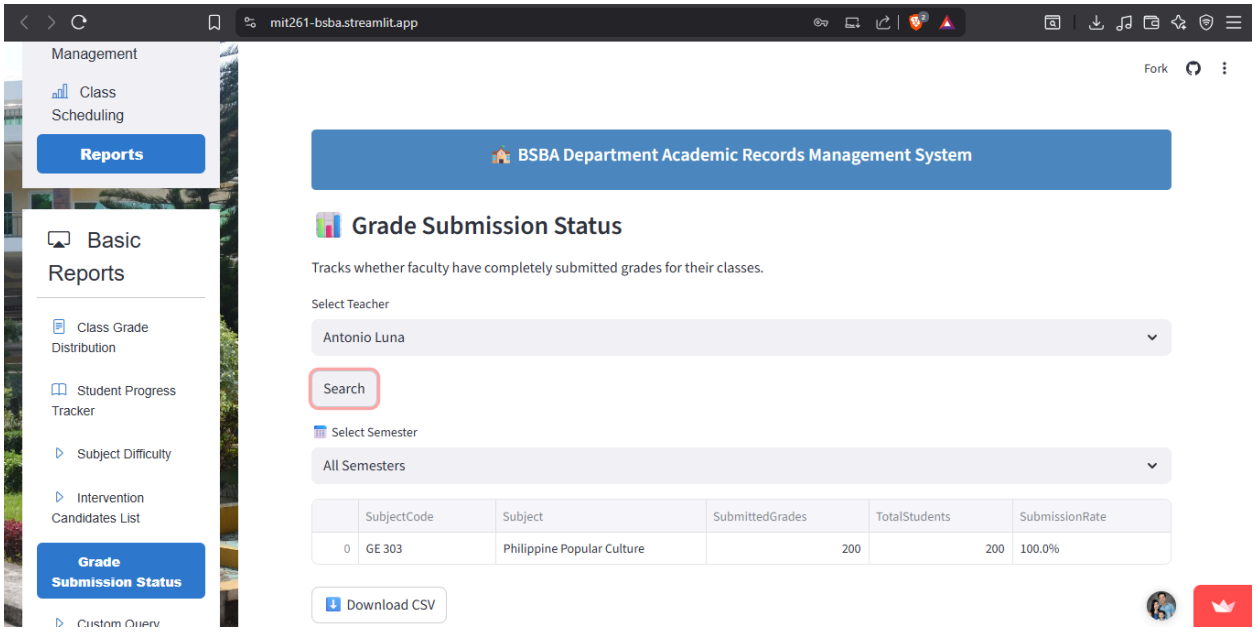
Faculty Name: Antonio Luna





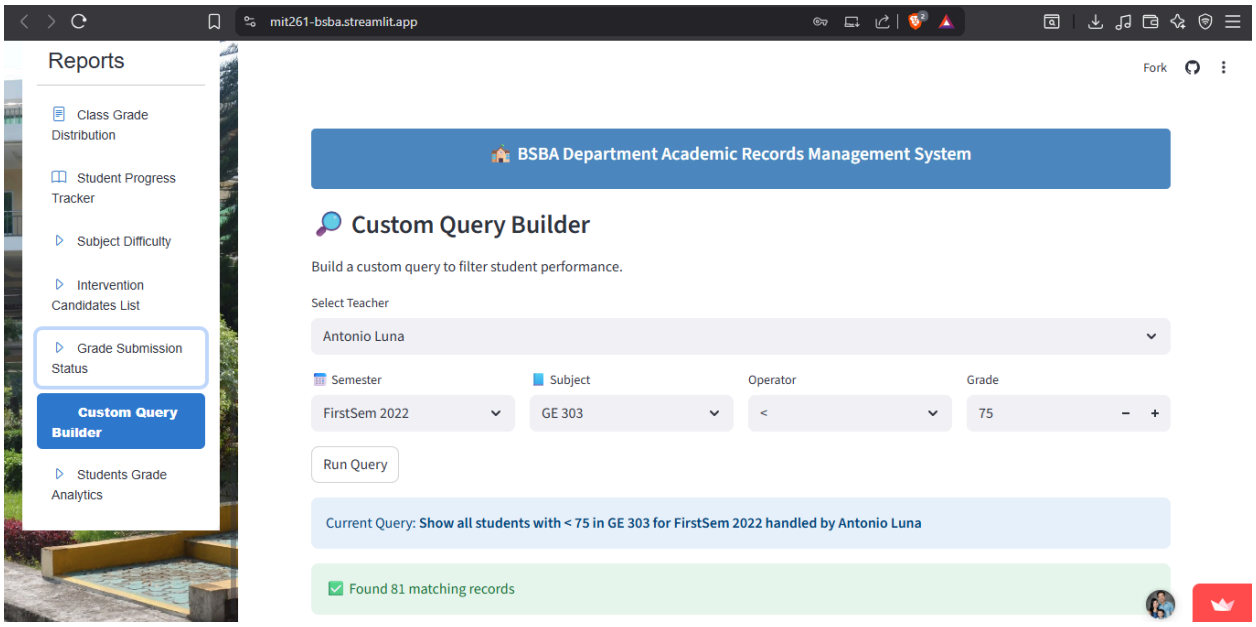
5. **Grade Submission Status**

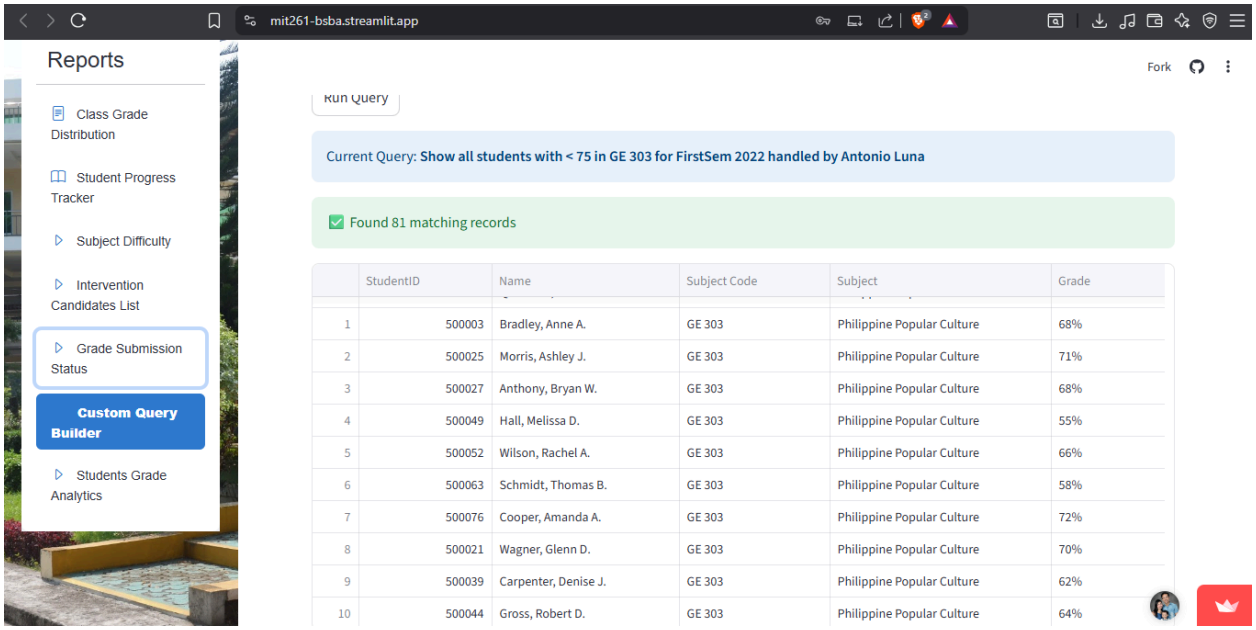
- **Tracks the status of grade submissions by faculty for each class. (e.g. complete grades, with blank grades)**



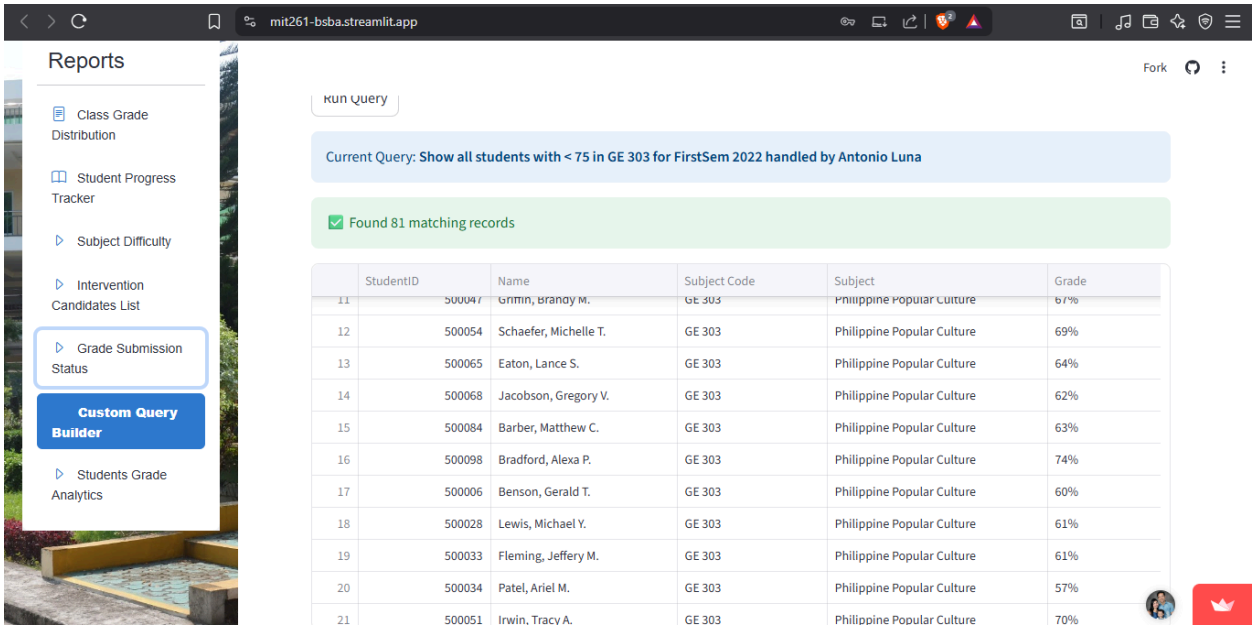
6. **Custom Query Builder**

- **Allows users to build filtered queries (e.g., "Show all students with <75 in CS101").**

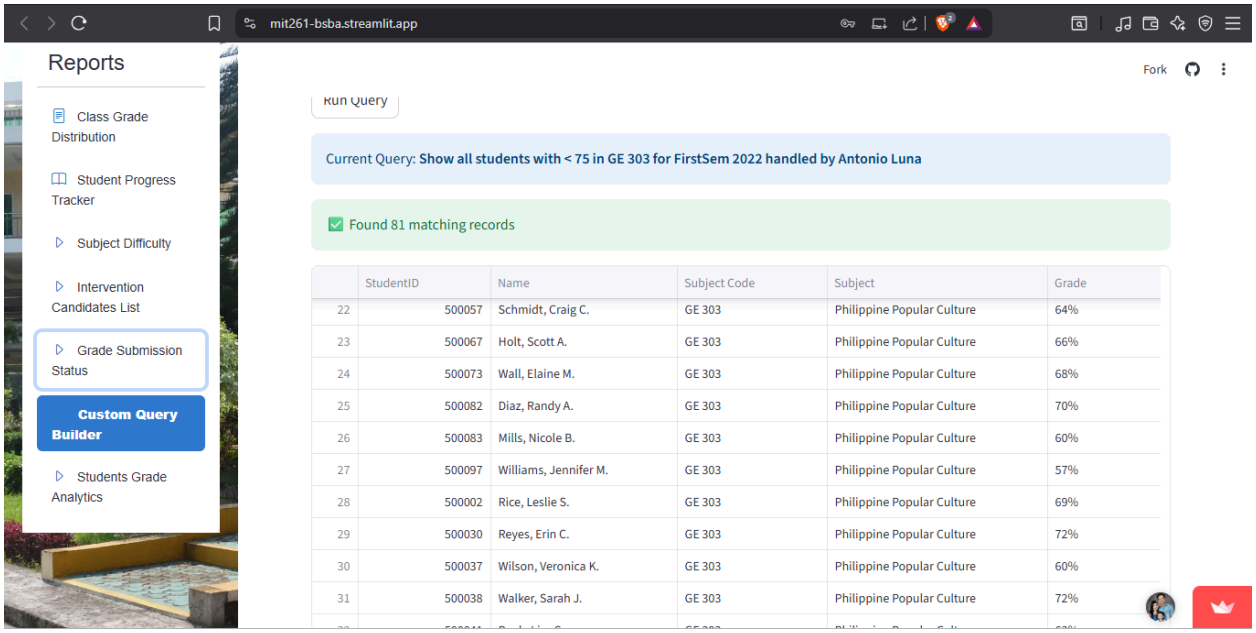




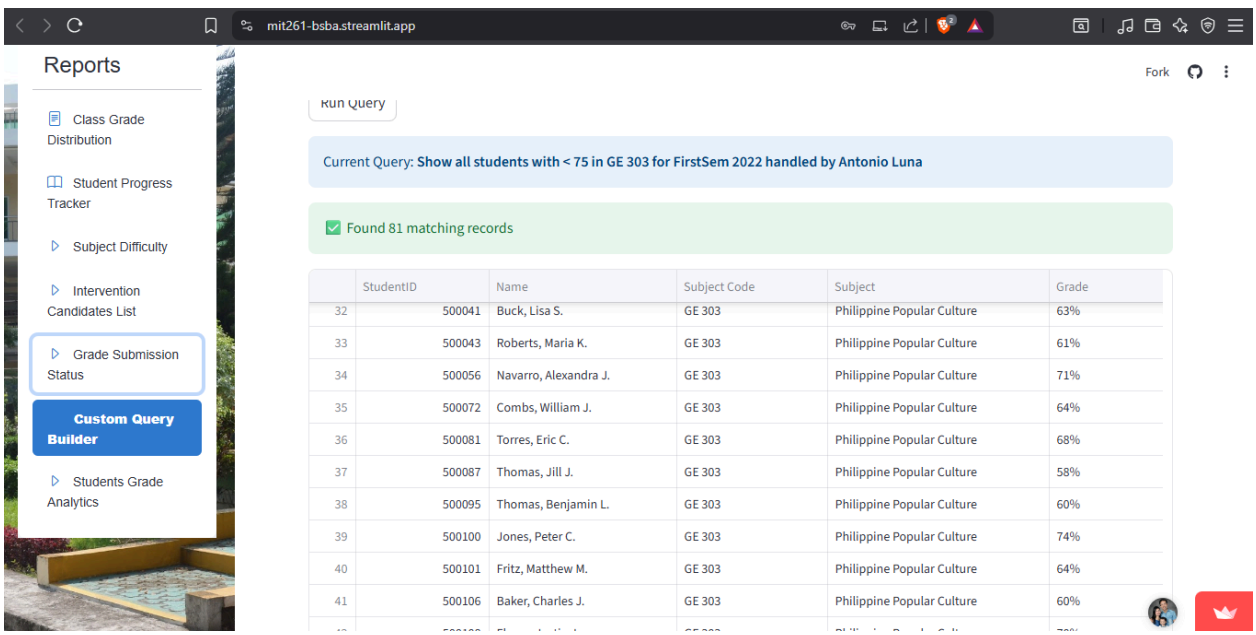
	StudentID	Name	Subject Code	Subject	Grade
1	500003	Bradley, Anne A.	GE 303	Philippine Popular Culture	68%
2	500025	Morris, Ashley J.	GE 303	Philippine Popular Culture	71%
3	500027	Anthony, Bryan W.	GE 303	Philippine Popular Culture	68%
4	500049	Hall, Melissa D.	GE 303	Philippine Popular Culture	55%
5	500052	Wilson, Rachel A.	GE 303	Philippine Popular Culture	66%
6	500063	Schmidt, Thomas B.	GE 303	Philippine Popular Culture	58%
7	500076	Cooper, Amanda A.	GE 303	Philippine Popular Culture	72%
8	500021	Wagner, Glenn D.	GE 303	Philippine Popular Culture	70%
9	500039	Carpenter, Denise J.	GE 303	Philippine Popular Culture	62%
10	500044	Gross, Robert D.	GE 303	Philippine Popular Culture	64%



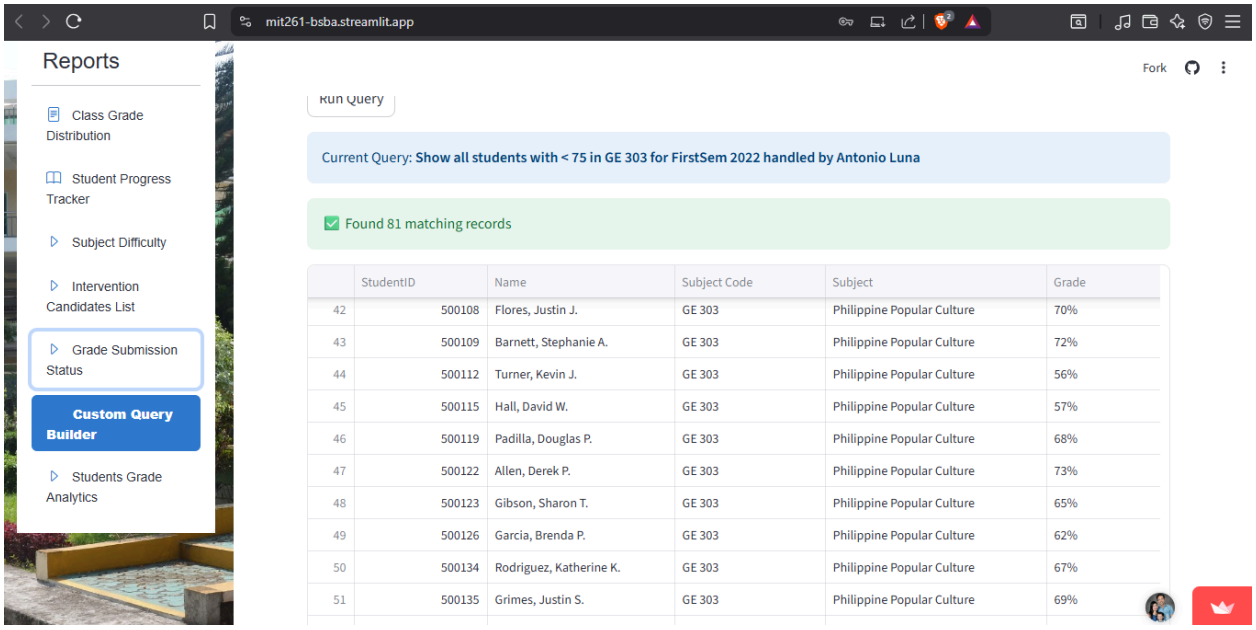
	StudentID	Name	Subject Code	Subject	Grade
11	500047	Griffin, Brandy M.	GE 303	Philippine Popular Culture	67%
12	500054	Schaefer, Michelle T.	GE 303	Philippine Popular Culture	69%
13	500065	Eaton, Lance S.	GE 303	Philippine Popular Culture	64%
14	500068	Jacobson, Gregory V.	GE 303	Philippine Popular Culture	62%
15	500084	Barber, Matthew C.	GE 303	Philippine Popular Culture	63%
16	500098	Bradford, Alexa P.	GE 303	Philippine Popular Culture	74%
17	500006	Benson, Gerald T.	GE 303	Philippine Popular Culture	60%
18	500028	Lewis, Michael Y.	GE 303	Philippine Popular Culture	61%
19	500033	Fleming, Jeffery M.	GE 303	Philippine Popular Culture	61%
20	500034	Patel, Ariel M.	GE 303	Philippine Popular Culture	57%
21	500051	Irwin, Tracy A.	GE 303	Philippine Popular Culture	70%

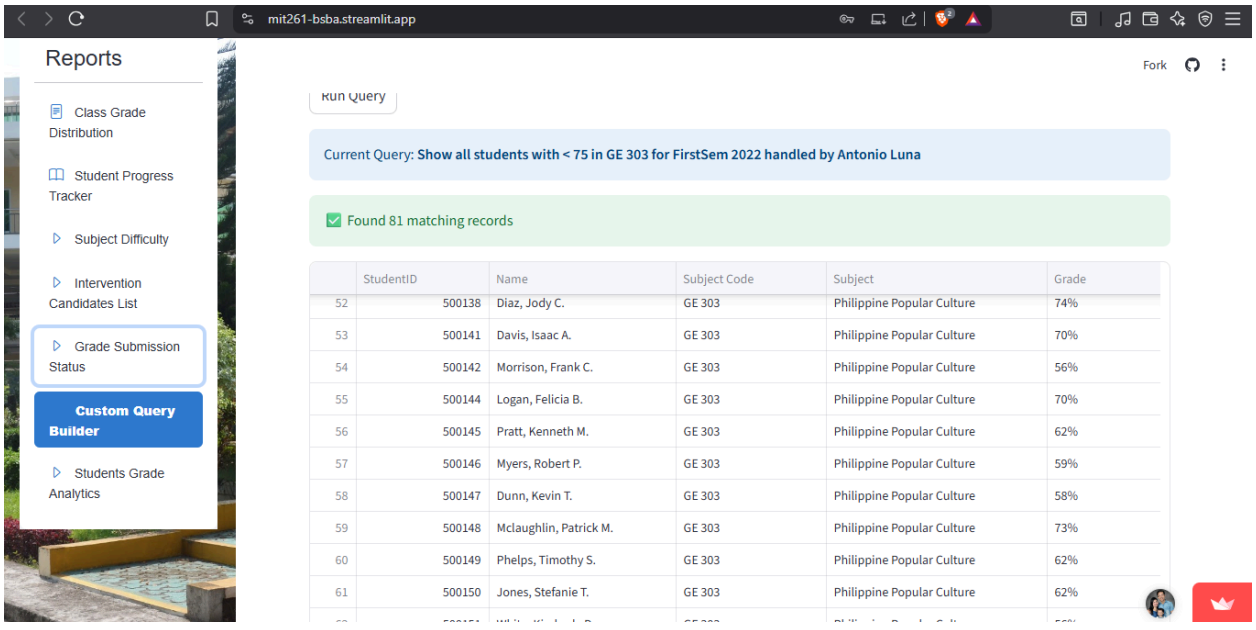


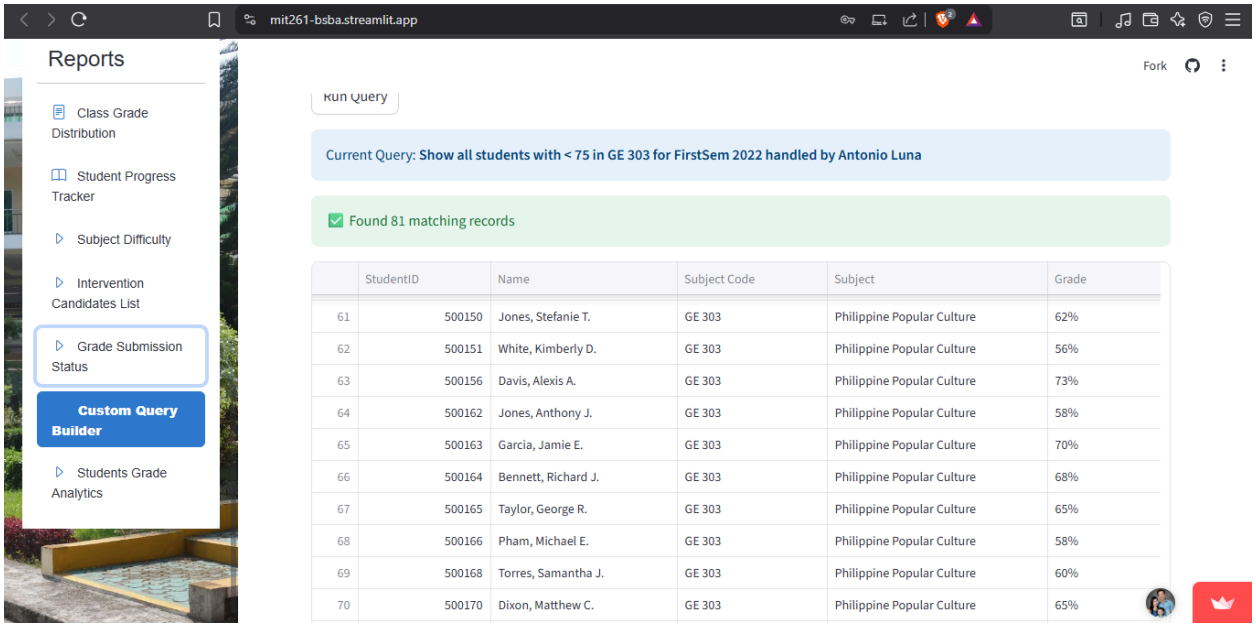
	StudentID	Name	Subject Code	Subject	Grade
22	500057	Schmidt, Craig C.	GE 303	Philippine Popular Culture	64%
23	500067	Holt, Scott A.	GE 303	Philippine Popular Culture	66%
24	500073	Wall, Elaine M.	GE 303	Philippine Popular Culture	68%
25	500082	Diaz, Randy A.	GE 303	Philippine Popular Culture	70%
26	500083	Mills, Nicole B.	GE 303	Philippine Popular Culture	60%
27	500097	Williams, Jennifer M.	GE 303	Philippine Popular Culture	57%
28	500002	Rice, Leslie S.	GE 303	Philippine Popular Culture	69%
29	500030	Reyes, Erin C.	GE 303	Philippine Popular Culture	72%
30	500037	Wilson, Veronica K.	GE 303	Philippine Popular Culture	60%
31	500038	Walker, Sarah J.	GE 303	Philippine Popular Culture	72%



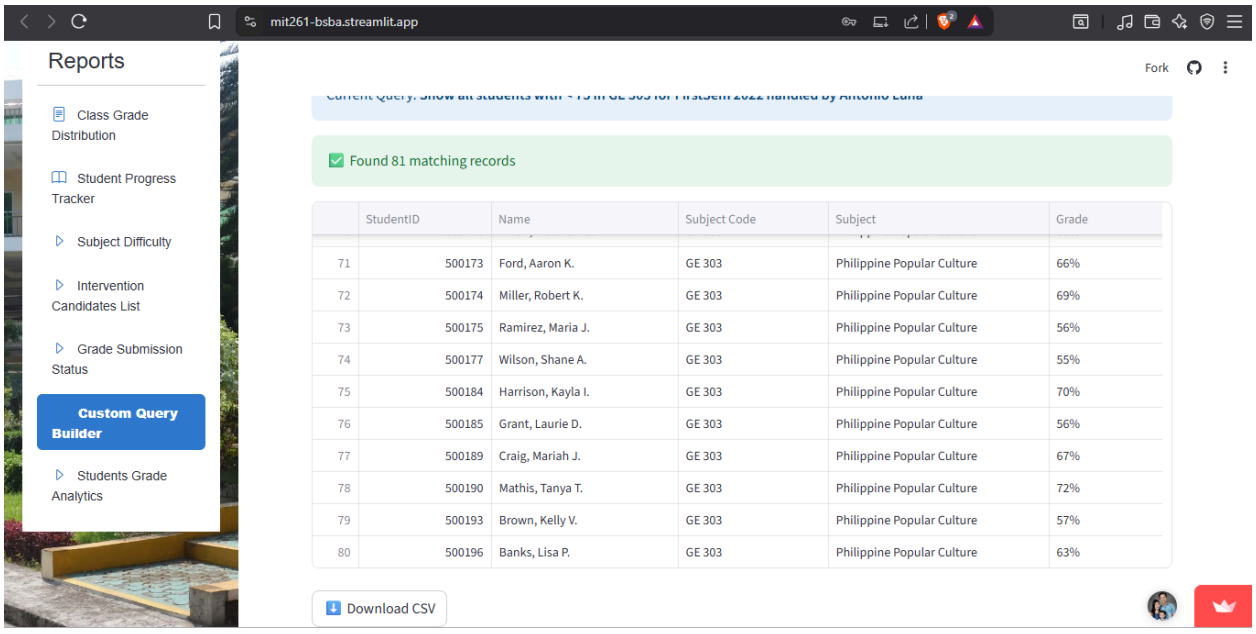
	StudentID	Name	Subject Code	Subject	Grade
32	500041	Buck, Lisa S.	GE 303	Philippine Popular Culture	63%
33	500043	Roberts, Maria K.	GE 303	Philippine Popular Culture	61%
34	500056	Navarro, Alexandra J.	GE 303	Philippine Popular Culture	71%
35	500072	Combs, William J.	GE 303	Philippine Popular Culture	64%
36	500081	Torres, Eric C.	GE 303	Philippine Popular Culture	68%
37	500087	Thomas, Jill J.	GE 303	Philippine Popular Culture	58%
38	500095	Thomas, Benjamin L.	GE 303	Philippine Popular Culture	60%
39	500100	Jones, Peter C.	GE 303	Philippine Popular Culture	74%
40	500101	Fritz, Matthew M.	GE 303	Philippine Popular Culture	64%
41	500106	Baker, Charles J.	GE 303	Philippine Popular Culture	60%


mit261-bsba.streamlit.app
Reports
Class Grade Distribution
Student Progress Tracker
Subject Difficulty
Intervention Candidates List
Grade Submission Status
Custom Query Builder
Students Grade Analytics
Fork
Run Query
Current Query: Show all students with < 75 in GE 303 for FirstSem 2022 handled by Antonio Luna
Found 81 matching records
StudentID Name Subject Code Subject Grade
42 500108 Flores, Justin J. GE 303 Philippine Popular Culture 70%
43 500109 Barnett, Stephanie A. GE 303 Philippine Popular Culture 72%
44 500112 Turner, Kevin J. GE 303 Philippine Popular Culture 56%
45 500115 Hall, David W. GE 303 Philippine Popular Culture 57%
46 500119 Padilla, Douglas P. GE 303 Philippine Popular Culture 68%
47 500122 Allen, Derek P. GE 303 Philippine Popular Culture 73%
48 500123 Gibson, Sharon T. GE 303 Philippine Popular Culture 65%
49 500126 Garcia, Brenda P. GE 303 Philippine Popular Culture 62%
50 500134 Rodriguez, Katherine K. GE 303 Philippine Popular Culture 67%
51 500135 Grimes, Justin S. GE 303 Philippine Popular Culture 69%


mit261-bsba.streamlit.app
Reports
Class Grade Distribution
Student Progress Tracker
Subject Difficulty
Intervention Candidates List
Grade Submission Status
Custom Query Builder
Students Grade Analytics
Fork
Run Query
Current Query: Show all students with < 75 in GE 303 for FirstSem 2022 handled by Antonio Luna
Found 81 matching records
StudentID Name Subject Code Subject Grade
52 500138 Diaz, Jody C. GE 303 Philippine Popular Culture 74%
53 500141 Davis, Isaac A. GE 303 Philippine Popular Culture 70%
54 500142 Morrison, Frank C. GE 303 Philippine Popular Culture 56%
55 500144 Logan, Felicia B. GE 303 Philippine Popular Culture 70%
56 500145 Pratt, Kenneth M. GE 303 Philippine Popular Culture 62%
57 500146 Myers, Robert P. GE 303 Philippine Popular Culture 59%
58 500147 Dunn, Kevin T. GE 303 Philippine Popular Culture 58%
59 500148 Mclaughlin, Patrick M. GE 303 Philippine Popular Culture 73%
60 500149 Phelps, Timothy S. GE 303 Philippine Popular Culture 62%
61 500150 Jones, Stefanie T. GE 303 Philippine Popular Culture 62%

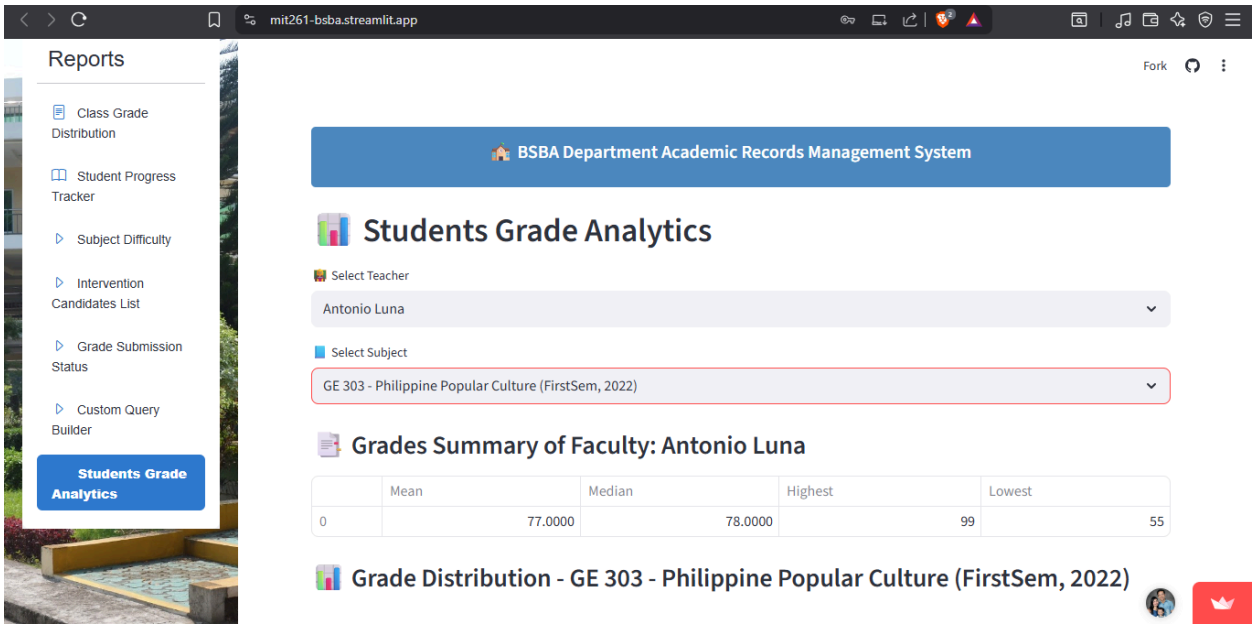


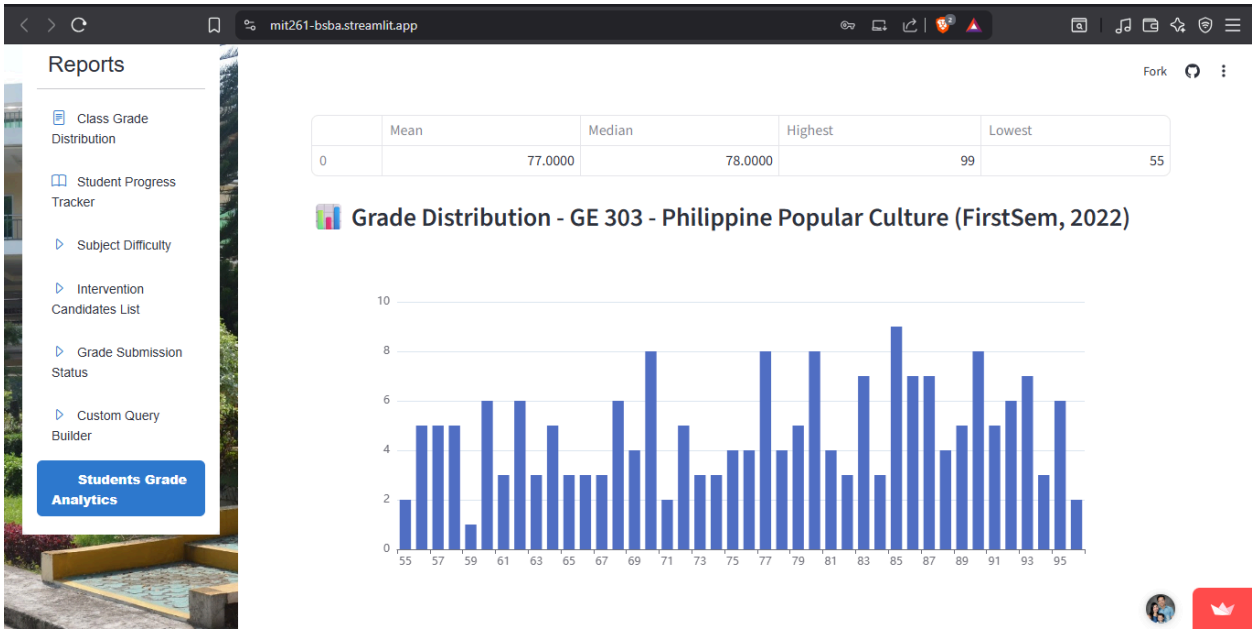
	StudentID	Name	Subject Code	Subject	Grade
61	500150	Jones, Stefanie T.	GE 303	Philippine Popular Culture	62%
62	500151	White, Kimberly D.	GE 303	Philippine Popular Culture	56%
63	500156	Davis, Alexis A.	GE 303	Philippine Popular Culture	73%
64	500162	Jones, Anthony J.	GE 303	Philippine Popular Culture	58%
65	500163	Garcia, Jamie E.	GE 303	Philippine Popular Culture	70%
66	500164	Bennett, Richard J.	GE 303	Philippine Popular Culture	68%
67	500165	Taylor, George R.	GE 303	Philippine Popular Culture	65%
68	500166	Pham, Michael E.	GE 303	Philippine Popular Culture	58%
69	500168	Torres, Samantha J.	GE 303	Philippine Popular Culture	60%
70	500170	Dixon, Matthew C.	GE 303	Philippine Popular Culture	65%

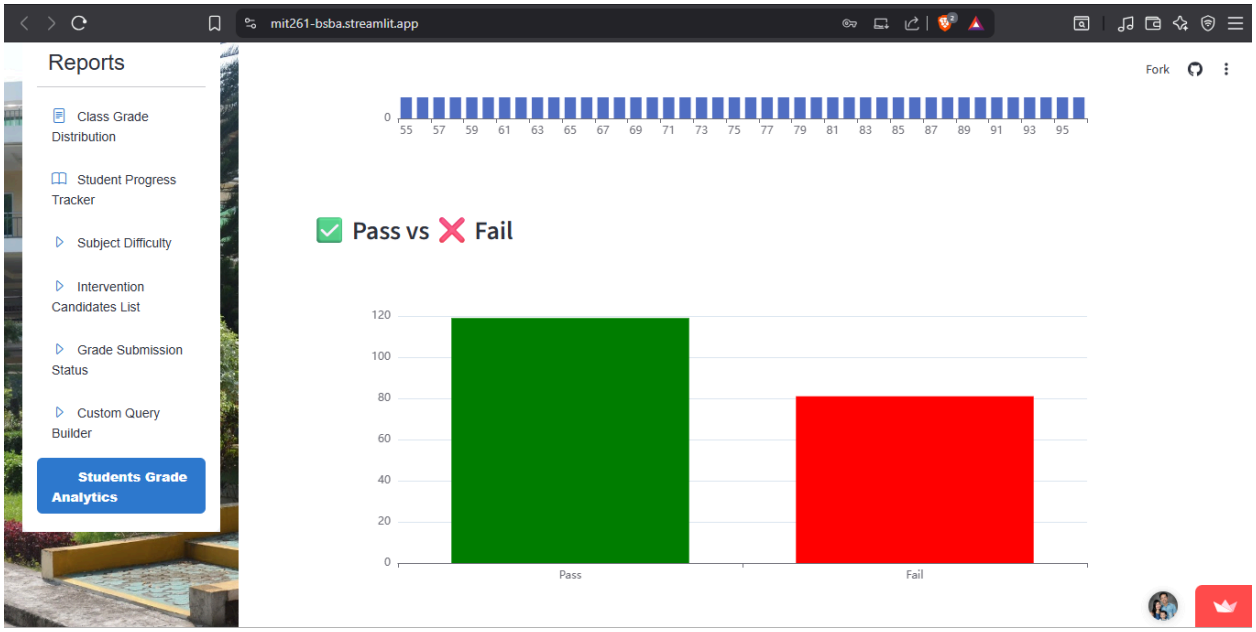


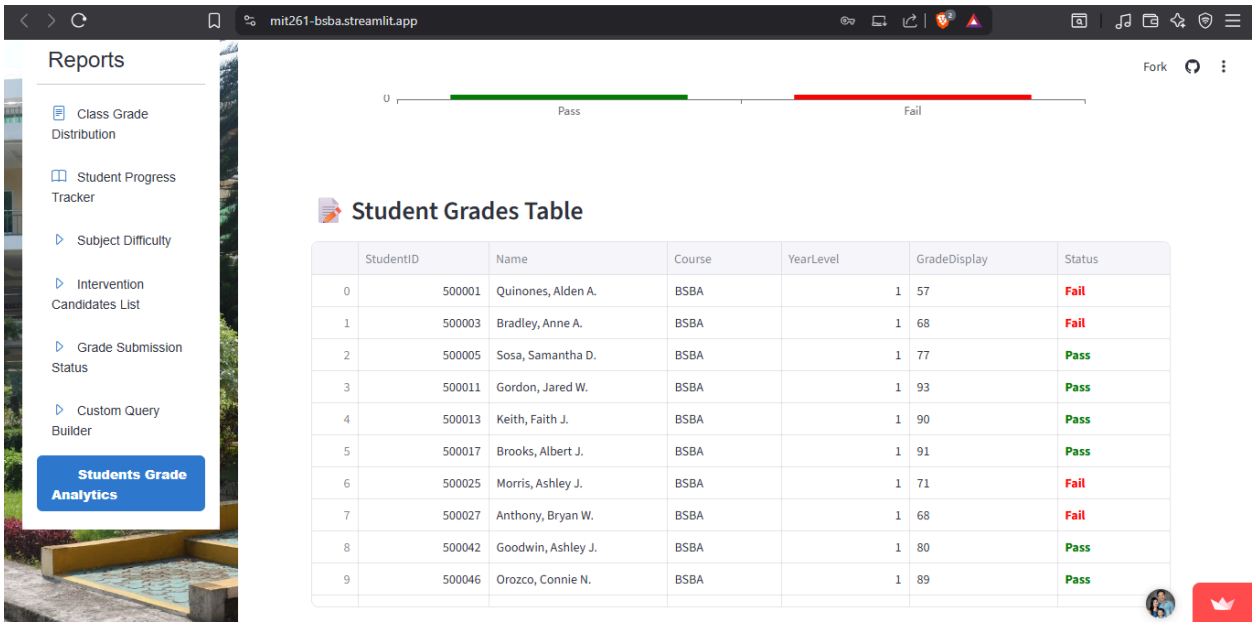
	StudentID	Name	Subject Code	Subject	Grade
71	500173	Ford, Aaron K.	GE 303	Philippine Popular Culture	66%
72	500174	Miller, Robert K.	GE 303	Philippine Popular Culture	69%
73	500175	Ramirez, Maria J.	GE 303	Philippine Popular Culture	56%
74	500177	Wilson, Shane A.	GE 303	Philippine Popular Culture	55%
75	500184	Harrison, Kayla I.	GE 303	Philippine Popular Culture	70%
76	500185	Grant, Laurie D.	GE 303	Philippine Popular Culture	56%
77	500189	Craig, Mariah J.	GE 303	Philippine Popular Culture	67%
78	500190	Mathis, Tanya T.	GE 303	Philippine Popular Culture	72%
79	500193	Brown, Kelly V.	GE 303	Philippine Popular Culture	57%
80	500196	Banks, Lisa P.	GE 303	Philippine Popular Culture	63%

7. Students Grade Analytics (Per Teacher)

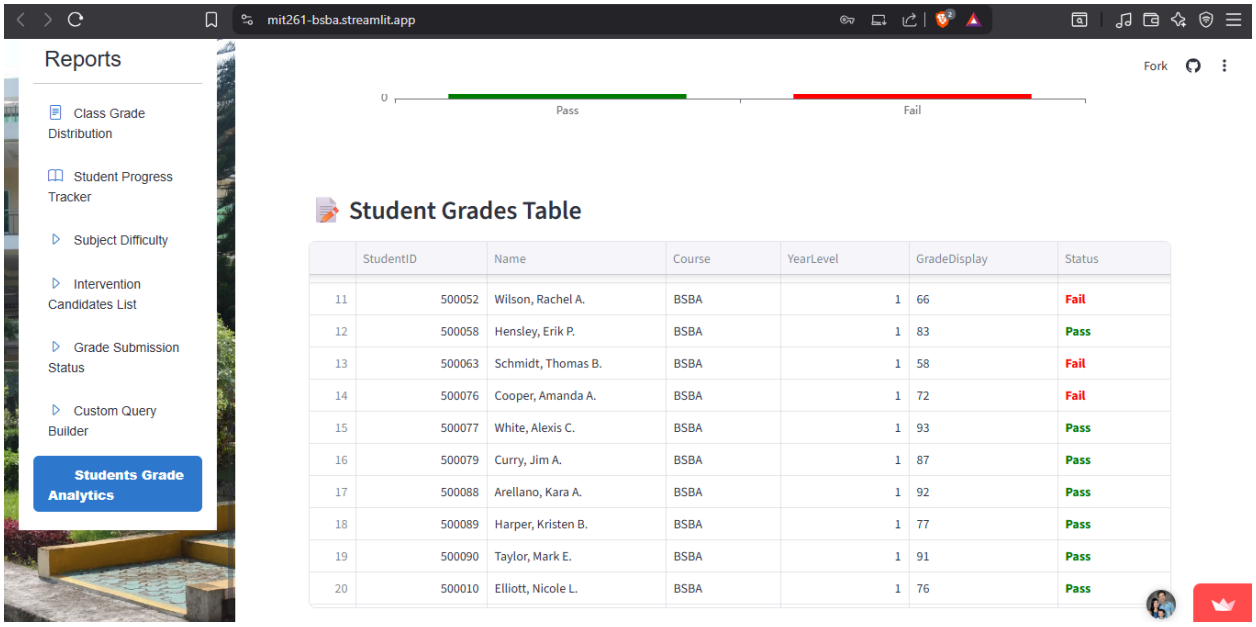




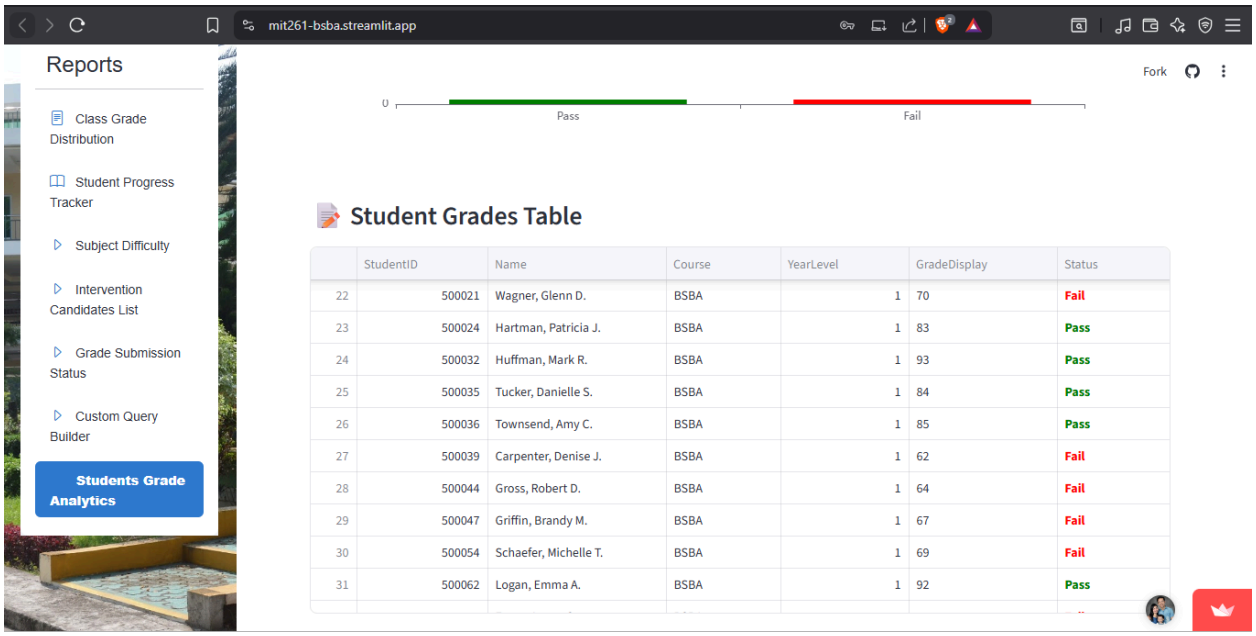




	StudentID	Name	Course	YearLevel	GradeDisplay	Status
0	500001	Quinones, Alden A.	BSBA	1	57	Fail
1	500003	Bradley, Anne A.	BSBA	1	68	Fail
2	500005	Sosa, Samantha D.	BSBA	1	77	Pass
3	500011	Gordon, Jared W.	BSBA	1	93	Pass
4	500013	Keith, Faith J.	BSBA	1	90	Pass
5	500017	Brooks, Albert J.	BSBA	1	91	Pass
6	500025	Morris, Ashley J.	BSBA	1	71	Fail
7	500027	Anthony, Bryan W.	BSBA	1	68	Fail
8	500042	Goodwin, Ashley J.	BSBA	1	80	Pass
9	500046	Orozco, Connie N.	BSBA	1	89	Pass



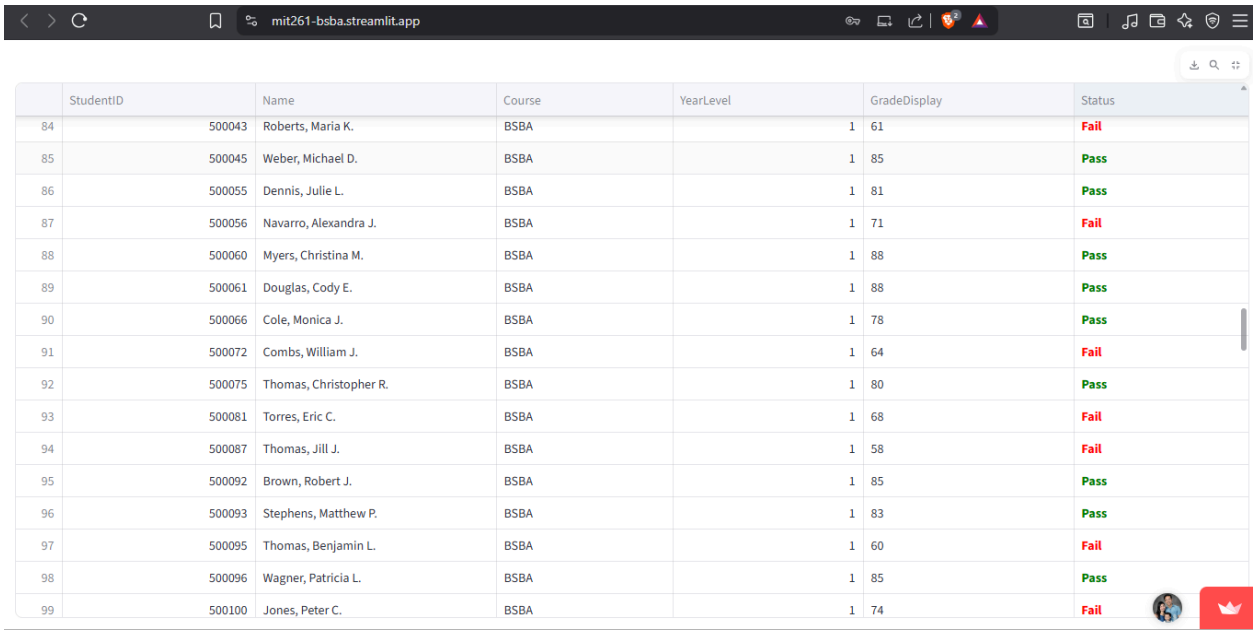
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
11	500052	Wilson, Rachel A.	BSBA	1	66	Fail
12	500058	Hensley, Erik P.	BSBA	1	83	Pass
13	500063	Schmidt, Thomas B.	BSBA	1	58	Fail
14	500076	Cooper, Amanda A.	BSBA	1	72	Fail
15	500077	White, Alexis C.	BSBA	1	93	Pass
16	500079	Curry, Jim A.	BSBA	1	87	Pass
17	500088	Arellano, Kara A.	BSBA	1	92	Pass
18	500089	Harper, Kristen B.	BSBA	1	77	Pass
19	500090	Taylor, Mark E.	BSBA	1	91	Pass
20	500010	Elliott, Nicole L.	BSBA	1	76	Pass



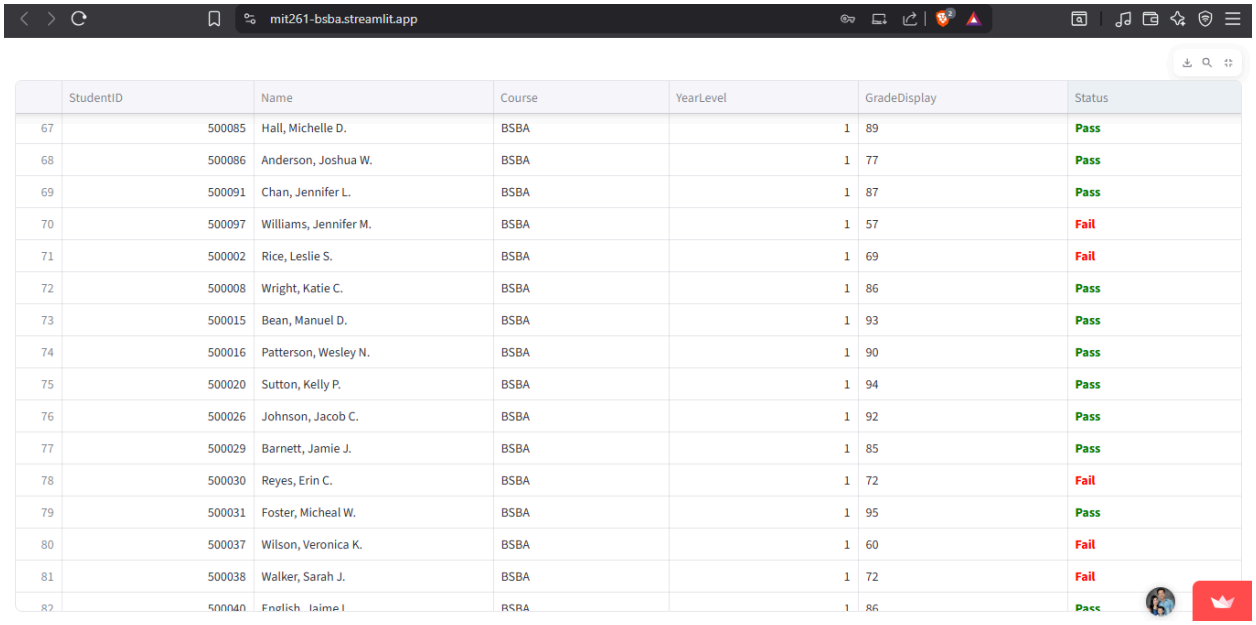
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
22	500021	Wagner, Glenn D.	BSBA	1	70	Fail
23	500024	Hartman, Patricia J.	BSBA	1	83	Pass
24	500032	Huffman, Mark R.	BSBA	1	93	Pass
25	500035	Tucker, Danielle S.	BSBA	1	84	Pass
26	500036	Townsend, Amy C.	BSBA	1	85	Pass
27	500039	Carpenter, Denise J.	BSBA	1	62	Fail
28	500044	Gross, Robert D.	BSBA	1	64	Fail
29	500047	Griffin, Brandy M.	BSBA	1	67	Fail
30	500054	Schaefer, Michelle T.	BSBA	1	69	Fail
31	500062	Logan, Emma A.	BSBA	1	92	Pass

	StudentID	Name	Course	YearLevel	GradeDisplay	Status
31	500062	Logan, Emma A.	BSBA	1	92	Pass
32	500065	Eaton, Lance S.	BSBA	1	64	Fail
33	500068	Jacobson, Gregory V.	BSBA	1	62	Fail
34	500069	Lloyd, Danielle A.	BSBA	1	76	Pass
35	500070	Villarreal, Shelley A.	BSBA	1	89	Pass
36	500074	Smith, Rachel D.	BSBA	1	82	Pass
37	500078	Fox, John J.	BSBA	1	93	Pass
38	500080	French, Andrew C.	BSBA	1	83	Pass
39	500084	Barber, Matthew C.	BSBA	1	63	Fail
40	500094	Williams, Anthony J.	BSBA	1	85	Pass
41	500098	Bradford, Alexa P.	BSBA	1	74	Fail
42	500099	Davis, Elizabeth D.	BSBA	1	79	Pass
43	500004	Little, Michael C.	BSBA	1	90	Pass
44	500006	Benson, Gerald T.	BSBA	1	60	Fail
45	500007	Green, Kevin A.	BSBA	1	87	Pass
46	500009	York, Tamara M.	BSBA	1	85	Pass

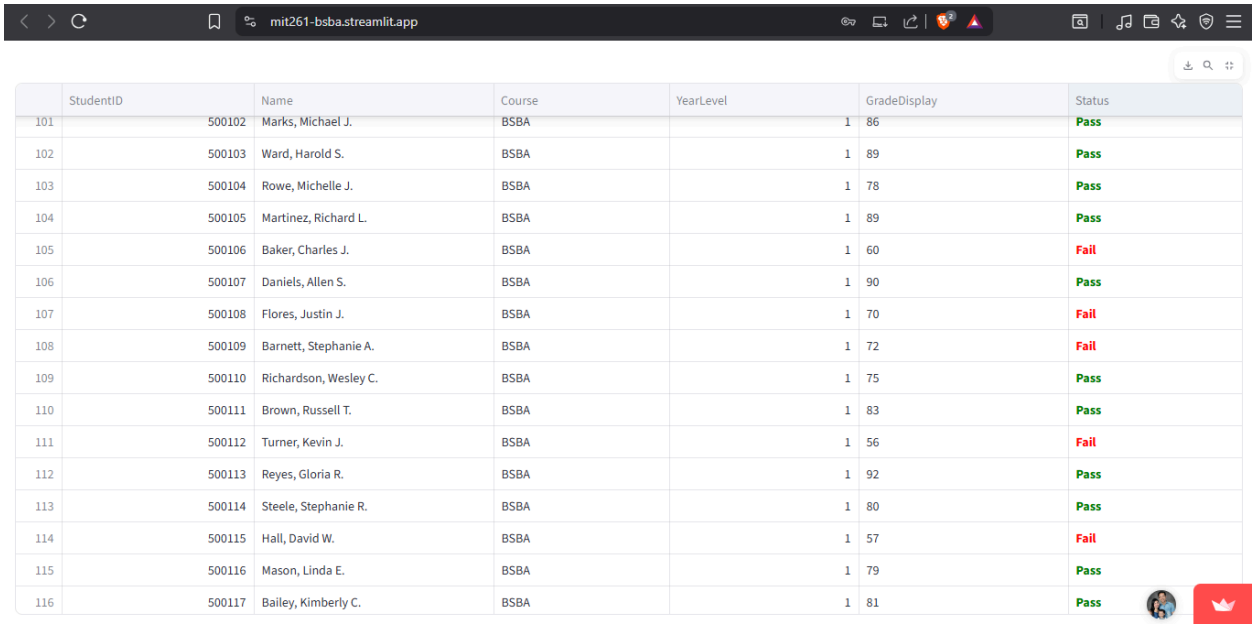
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
47	500012	Silva, Robert D.	BSBA	1	93	Pass
48	500014	Johnson, Jennifer C.	BSBA	1	90	Pass
49	500018	Schaefer, Kayla W.	BSBA	1	99	Pass
50	500022	Brown, Eric G.	BSBA	1	77	Pass
51	500023	Massey, Anthony W.	BSBA	1	91	Pass
52	500028	Lewis, Michael Y.	BSBA	1	61	Fail
53	500033	Fleming, Jeffery M.	BSBA	1	61	Fail
54	500034	Patel, Ariel M.	BSBA	1	57	Fail
55	500048	Williams, Gary M.	BSBA	1	86	Pass
56	500050	Cabrera, Kevin L.	BSBA	1	83	Pass
57	500051	Irwin, Tracy A.	BSBA	1	70	Fail
58	500053	Rosario, Zachary L.	BSBA	1	95	Pass
59	500057	Schmidt, Craig C.	BSBA	1	64	Fail
60	500059	Williams, Linda R.	BSBA	1	88	Pass
61	500064	Marquez, Melissa B.	BSBA	1	75	Pass
62	500067	Holt, Scott A.	BSBA	1	66	Fail



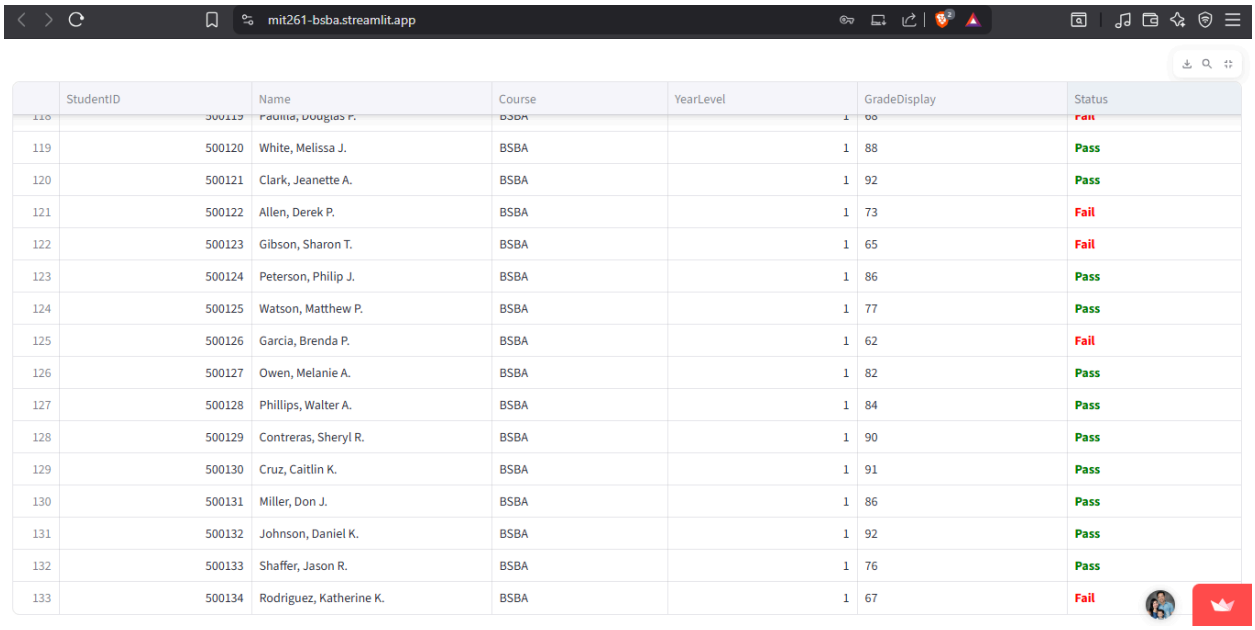
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
84	500043	Roberts, Maria K.	BSBA	1	61	Fail
85	500045	Weber, Michael D.	BSBA	1	85	Pass
86	500055	Dennis, Julie L.	BSBA	1	81	Pass
87	500056	Navarro, Alexandra J.	BSBA	1	71	Fail
88	500060	Myers, Christina M.	BSBA	1	88	Pass
89	500061	Douglas, Cody E.	BSBA	1	88	Pass
90	500066	Cole, Monica J.	BSBA	1	78	Pass
91	500072	Combs, William J.	BSBA	1	64	Fail
92	500075	Thomas, Christopher R.	BSBA	1	80	Pass
93	500081	Torres, Eric C.	BSBA	1	68	Fail
94	500087	Thomas, Jill J.	BSBA	1	58	Fail
95	500092	Brown, Robert J.	BSBA	1	85	Pass
96	500093	Stephens, Matthew P.	BSBA	1	83	Pass
97	500095	Thomas, Benjamin L.	BSBA	1	60	Fail
98	500096	Wagner, Patricia L.	BSBA	1	85	Pass
99	500100	Jones, Peter C.	BSBA	1	74	Fail



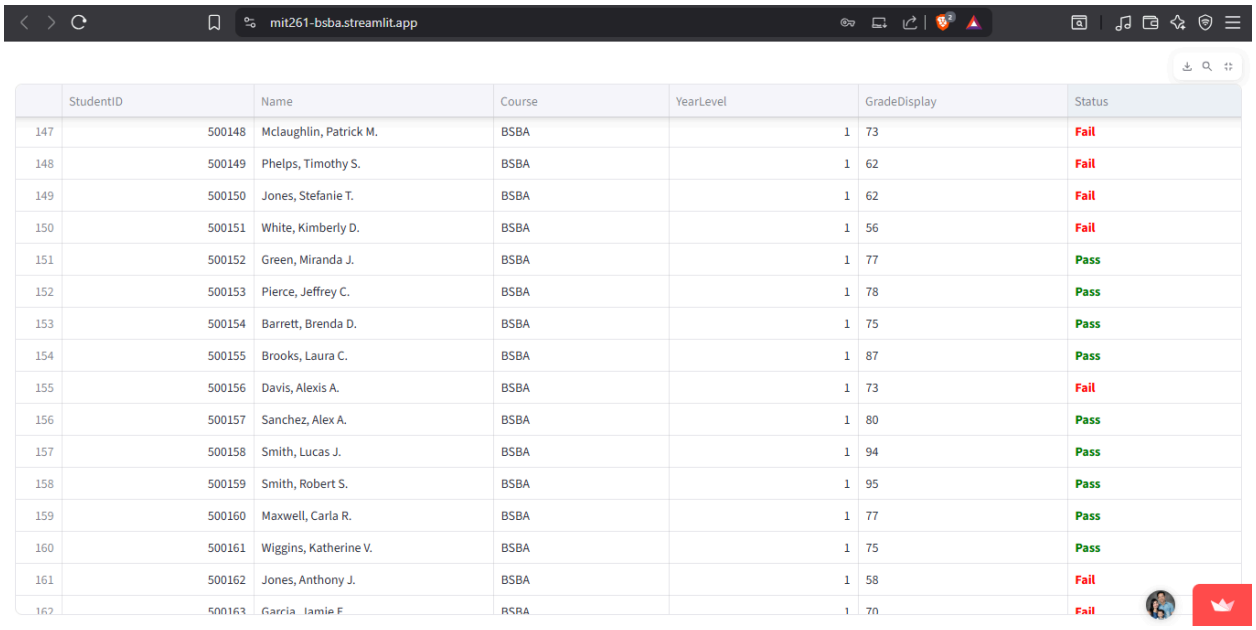
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
67	500085	Hall, Michelle D.	BSBA	1	89	Pass
68	500086	Anderson, Joshua W.	BSBA	1	77	Pass
69	500091	Chan, Jennifer L.	BSBA	1	87	Pass
70	500097	Williams, Jennifer M.	BSBA	1	57	Fail
71	500002	Rice, Leslie S.	BSBA	1	69	Fail
72	500008	Wright, Katie C.	BSBA	1	86	Pass
73	500015	Bean, Manuel D.	BSBA	1	93	Pass
74	500016	Patterson, Wesley N.	BSBA	1	90	Pass
75	500020	Sutton, Kelly P.	BSBA	1	94	Pass
76	500026	Johnson, Jacob C.	BSBA	1	92	Pass
77	500029	Barnett, Jamie J.	BSBA	1	85	Pass
78	500030	Reyes, Erin C.	BSBA	1	72	Fail
79	500031	Foster, Micheal W.	BSBA	1	95	Pass
80	500037	Wilson, Veronica K.	BSBA	1	60	Fail
81	500038	Walker, Sarah J.	BSBA	1	72	Fail
82	500040	English, Jaime L.	BSBA	1	86	Pass



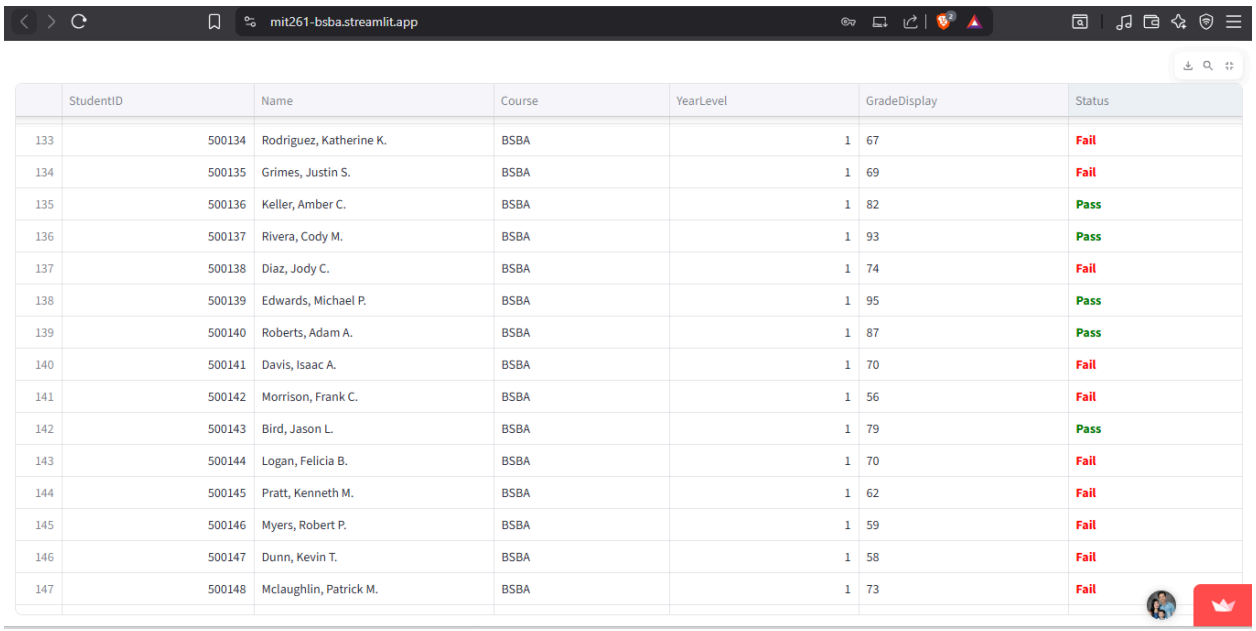
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
101	500102	Marks, Michael J.	BSBA	1	86	Pass
102	500103	Ward, Harold S.	BSBA	1	89	Pass
103	500104	Rowe, Michelle J.	BSBA	1	78	Pass
104	500105	Martinez, Richard L.	BSBA	1	89	Pass
105	500106	Baker, Charles J.	BSBA	1	60	Fail
106	500107	Daniels, Allen S.	BSBA	1	90	Pass
107	500108	Flores, Justin J.	BSBA	1	70	Fail
108	500109	Barnett, Stephanie A.	BSBA	1	72	Fail
109	500110	Richardson, Wesley C.	BSBA	1	75	Pass
110	500111	Brown, Russell T.	BSBA	1	83	Pass
111	500112	Turner, Kevin J.	BSBA	1	56	Fail
112	500113	Reyes, Gloria R.	BSBA	1	92	Pass
113	500114	Steele, Stephanie R.	BSBA	1	80	Pass
114	500115	Hall, David W.	BSBA	1	57	Fail
115	500116	Mason, Linda E.	BSBA	1	79	Pass
116	500117	Bailey, Kimberly C.	BSBA	1	81	Pass



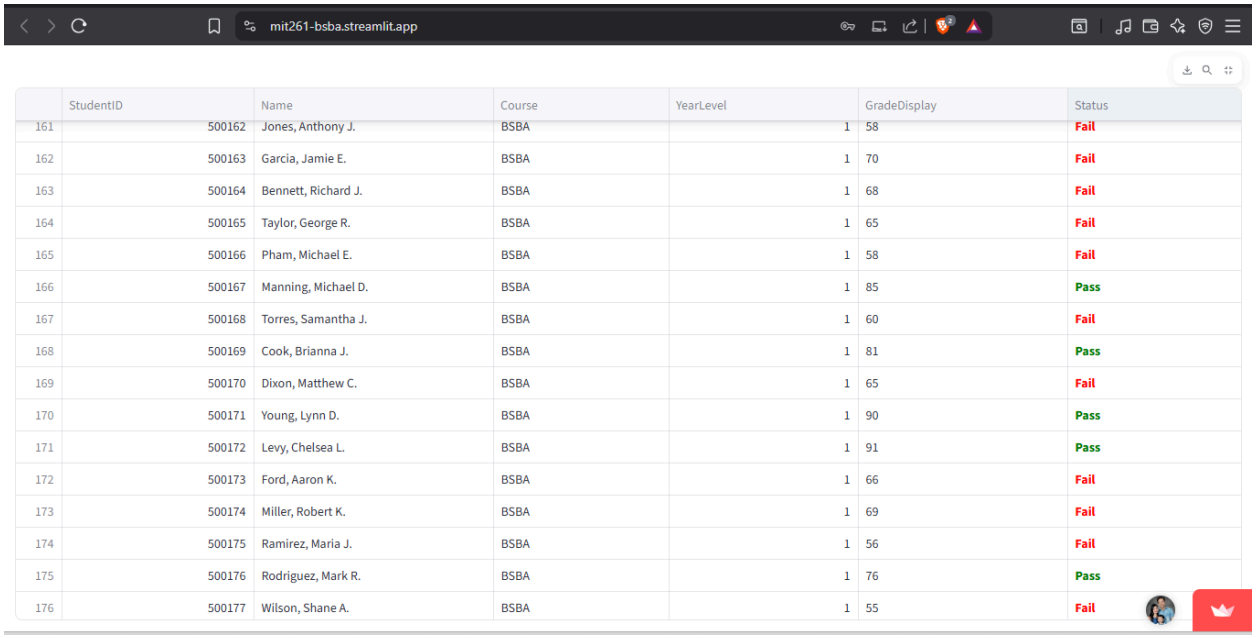
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
118	500119	Padilla, Douglas P.	BSBA	1	68	Fail
119	500120	White, Melissa J.	BSBA	1	88	Pass
120	500121	Clark, Jeanette A.	BSBA	1	92	Pass
121	500122	Allen, Derek P.	BSBA	1	73	Fail
122	500123	Gibson, Sharon T.	BSBA	1	65	Fail
123	500124	Peterson, Philip J.	BSBA	1	86	Pass
124	500125	Watson, Matthew P.	BSBA	1	77	Pass
125	500126	Garcia, Brenda P.	BSBA	1	62	Fail
126	500127	Owen, Melanie A.	BSBA	1	82	Pass
127	500128	Phillips, Walter A.	BSBA	1	84	Pass
128	500129	Contreras, Sheryl R.	BSBA	1	90	Pass
129	500130	Cruz, Caitlin K.	BSBA	1	91	Pass
130	500131	Miller, Don J.	BSBA	1	86	Pass
131	500132	Johnson, Daniel K.	BSBA	1	92	Pass
132	500133	Shaffer, Jason R.	BSBA	1	76	Pass
133	500134	Rodriguez, Katherine K.	BSBA	1	67	Fail



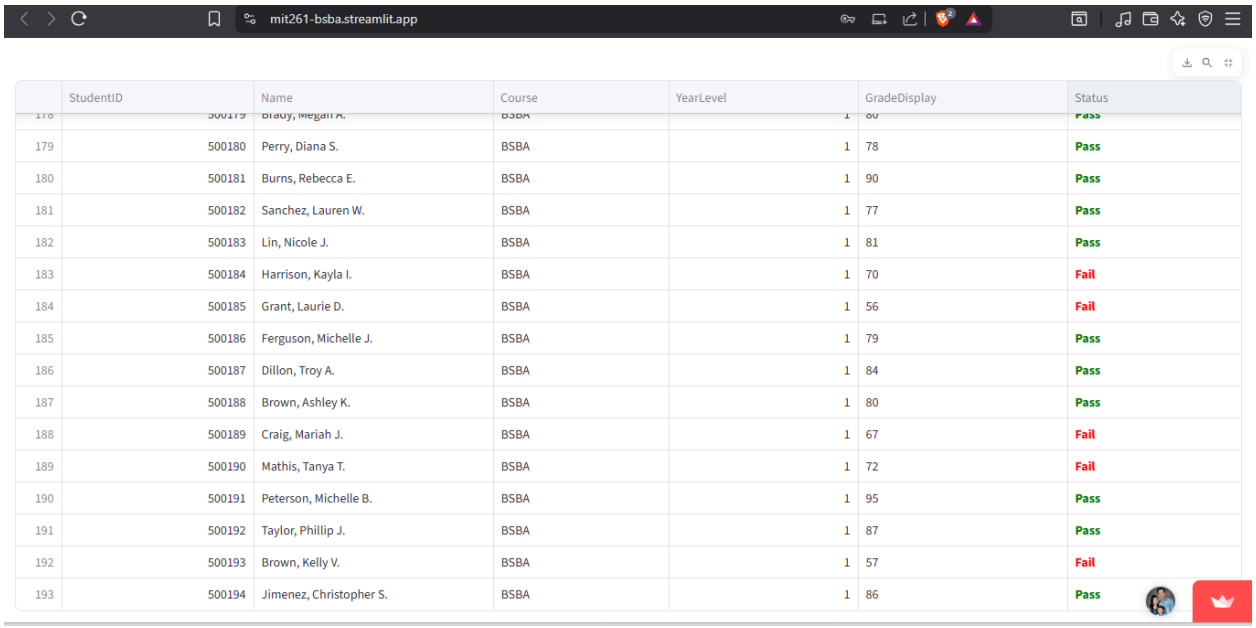
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
147	500148	Mclaughlin, Patrick M.	BSBA	1	73	Fail
148	500149	Phelps, Timothy S.	BSBA	1	62	Fail
149	500150	Jones, Stefanie T.	BSBA	1	62	Fail
150	500151	White, Kimberly D.	BSBA	1	56	Fail
151	500152	Green, Miranda J.	BSBA	1	77	Pass
152	500153	Pierce, Jeffrey C.	BSBA	1	78	Pass
153	500154	Barrett, Brenda D.	BSBA	1	75	Pass
154	500155	Brooks, Laura C.	BSBA	1	87	Pass
155	500156	Davis, Alexis A.	BSBA	1	73	Fail
156	500157	Sanchez, Alex A.	BSBA	1	80	Pass
157	500158	Smith, Lucas J.	BSBA	1	94	Pass
158	500159	Smith, Robert S.	BSBA	1	95	Pass
159	500160	Maxwell, Carla R.	BSBA	1	77	Pass
160	500161	Wiggins, Katherine V.	BSBA	1	75	Pass
161	500162	Jones, Anthony J.	BSBA	1	58	Fail
162	500163	Garcia, Jamie E.	BSBA	1	70	Fail



	StudentID	Name	Course	YearLevel	GradeDisplay	Status
133	500134	Rodriguez, Katherine K.	BSBA	1	67	Fail
134	500135	Grimes, Justin S.	BSBA	1	69	Fail
135	500136	Keller, Amber C.	BSBA	1	82	Pass
136	500137	Rivera, Cody M.	BSBA	1	93	Pass
137	500138	Diaz, Jody C.	BSBA	1	74	Fail
138	500139	Edwards, Michael P.	BSBA	1	95	Pass
139	500140	Roberts, Adam A.	BSBA	1	87	Pass
140	500141	Davis, Isaac A.	BSBA	1	70	Fail
141	500142	Morrison, Frank C.	BSBA	1	56	Fail
142	500143	Bird, Jason L.	BSBA	1	79	Pass
143	500144	Logan, Felicia B.	BSBA	1	70	Fail
144	500145	Pratt, Kenneth M.	BSBA	1	62	Fail
145	500146	Myers, Robert P.	BSBA	1	59	Fail
146	500147	Dunn, Kevin T.	BSBA	1	58	Fail
147	500148	Mclaughlin, Patrick M.	BSBA	1	73	Fail



	StudentID	Name	Course	YearLevel	GradeDisplay	Status
161	500162	Jones, Anthony J.	BSBA	1	58	Fail
162	500163	Garcia, Jamie E.	BSBA	1	70	Fail
163	500164	Bennett, Richard J.	BSBA	1	68	Fail
164	500165	Taylor, George R.	BSBA	1	65	Fail
165	500166	Pham, Michael E.	BSBA	1	58	Fail
166	500167	Manning, Michael D.	BSBA	1	85	Pass
167	500168	Torres, Samantha J.	BSBA	1	60	Fail
168	500169	Cook, Brianna J.	BSBA	1	81	Pass
169	500170	Dixon, Matthew C.	BSBA	1	65	Fail
170	500171	Young, Lynn D.	BSBA	1	90	Pass
171	500172	Levy, Chelsea L.	BSBA	1	91	Pass
172	500173	Ford, Aaron K.	BSBA	1	66	Fail
173	500174	Miller, Robert K.	BSBA	1	69	Fail
174	500175	Ramirez, Maria J.	BSBA	1	56	Fail
175	500176	Rodriguez, Mark R.	BSBA	1	76	Pass
176	500177	Wilson, Shane A.	BSBA	1	55	Fail

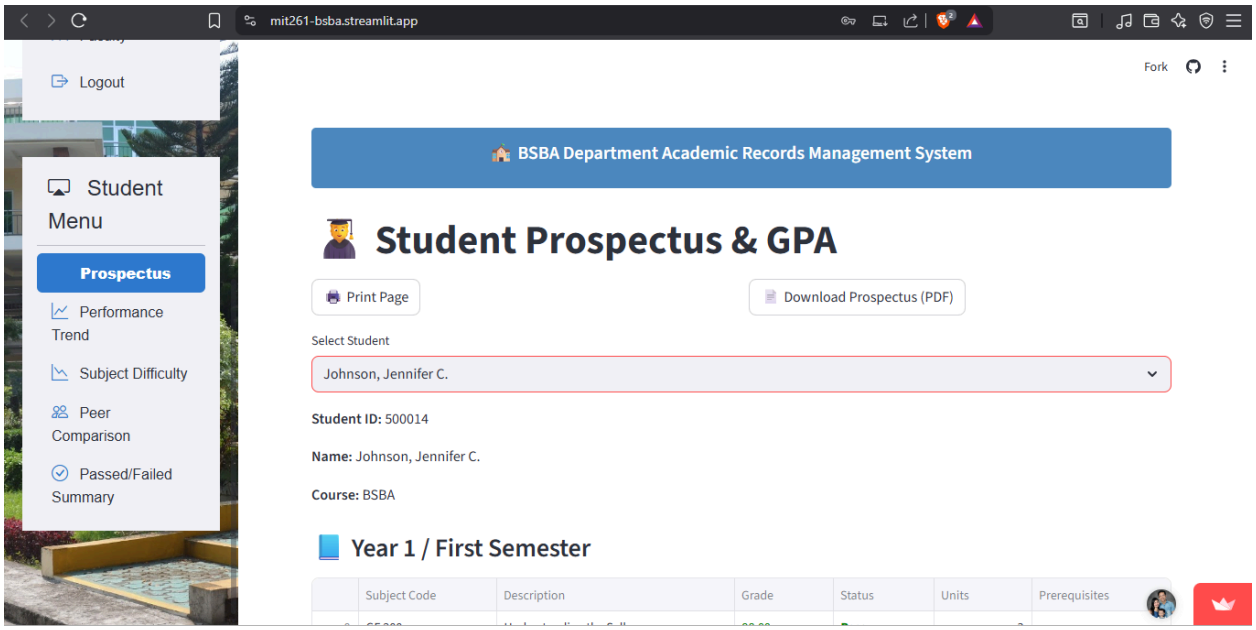


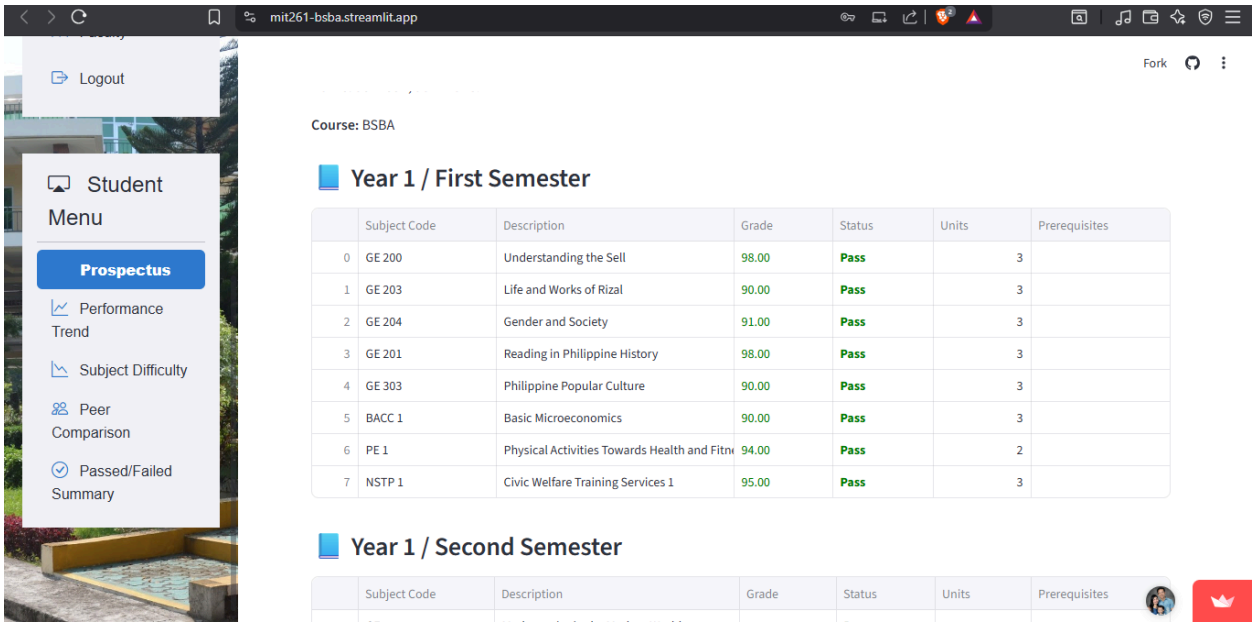
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
178	500179	Brady, Megan A.	BSBA	1	80	Pass
179	500180	Perry, Diana S.	BSBA	1	78	Pass
180	500181	Burns, Rebecca E.	BSBA	1	90	Pass
181	500182	Sanchez, Lauren W.	BSBA	1	77	Pass
182	500183	Lin, Nicole J.	BSBA	1	81	Pass
183	500184	Harrison, Kayla I.	BSBA	1	70	Fail
184	500185	Grant, Laurie D.	BSBA	1	56	Fail
185	500186	Ferguson, Michelle J.	BSBA	1	79	Pass
186	500187	Dillon, Troy A.	BSBA	1	84	Pass
187	500188	Brown, Ashley K.	BSBA	1	80	Pass
188	500189	Craig, Mariah J.	BSBA	1	67	Fail
189	500190	Mathis, Tanya T.	BSBA	1	72	Fail
190	500191	Peterson, Michelle B.	BSBA	1	95	Pass
191	500192	Taylor, Phillip J.	BSBA	1	87	Pass
192	500193	Brown, Kelly V.	BSBA	1	57	Fail
193	500194	Jimenez, Christopher S.	BSBA	1	86	Pass

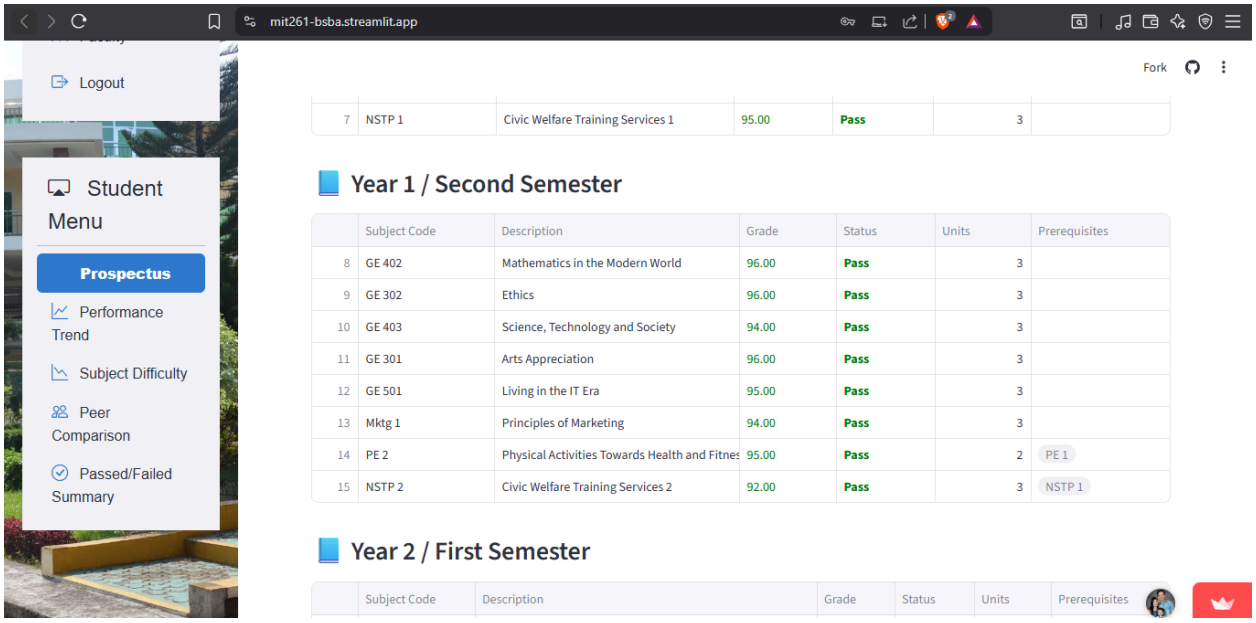
	StudentID	Name	Course	YearLevel	GradeDisplay	Status
184	500185	Grant, Laurie D.	BSBA	1	56	Fail
185	500186	Ferguson, Michelle J.	BSBA	1	79	Pass
186	500187	Dillon, Troy A.	BSBA	1	84	Pass
187	500188	Brown, Ashley K.	BSBA	1	80	Pass
188	500189	Craig, Mariah J.	BSBA	1	67	Fail
189	500190	Mathis, Tanya T.	BSBA	1	72	Fail
190	500191	Peterson, Michelle B.	BSBA	1	95	Pass
191	500192	Taylor, Phillip J.	BSBA	1	87	Pass
192	500193	Brown, Kelly V.	BSBA	1	57	Fail
193	500194	Jimenez, Christopher S.	BSBA	1	86	Pass
194	500195	Boyd, Morgan A.	BSBA	1	83	Pass
195	500196	Banks, Lisa P.	BSBA	1	63	Fail
196	500197	Anderson, Danny L.	BSBA	1	80	Pass
197	500198	Garcia, Amy J.	BSBA	1	85	Pass
198	500199	Kaufman, William M.	BSBA	1	79	Pass
199	500200	Garcia, Matthew D.	BSBA	1	80	Pass

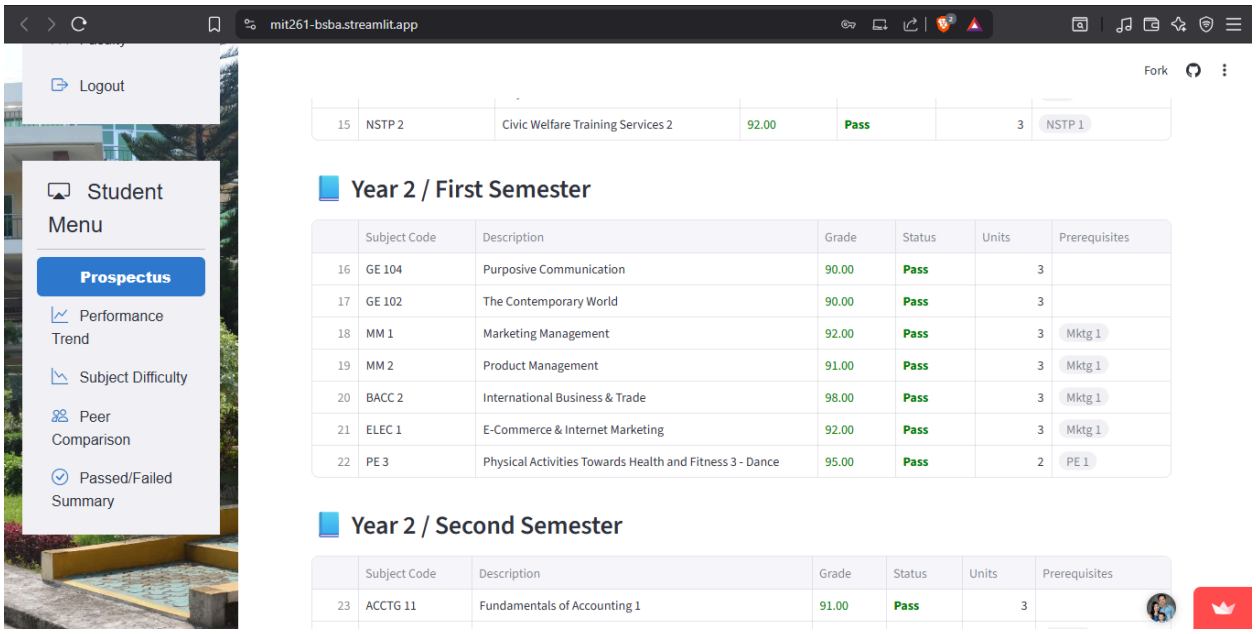
Students Dashboard Reports

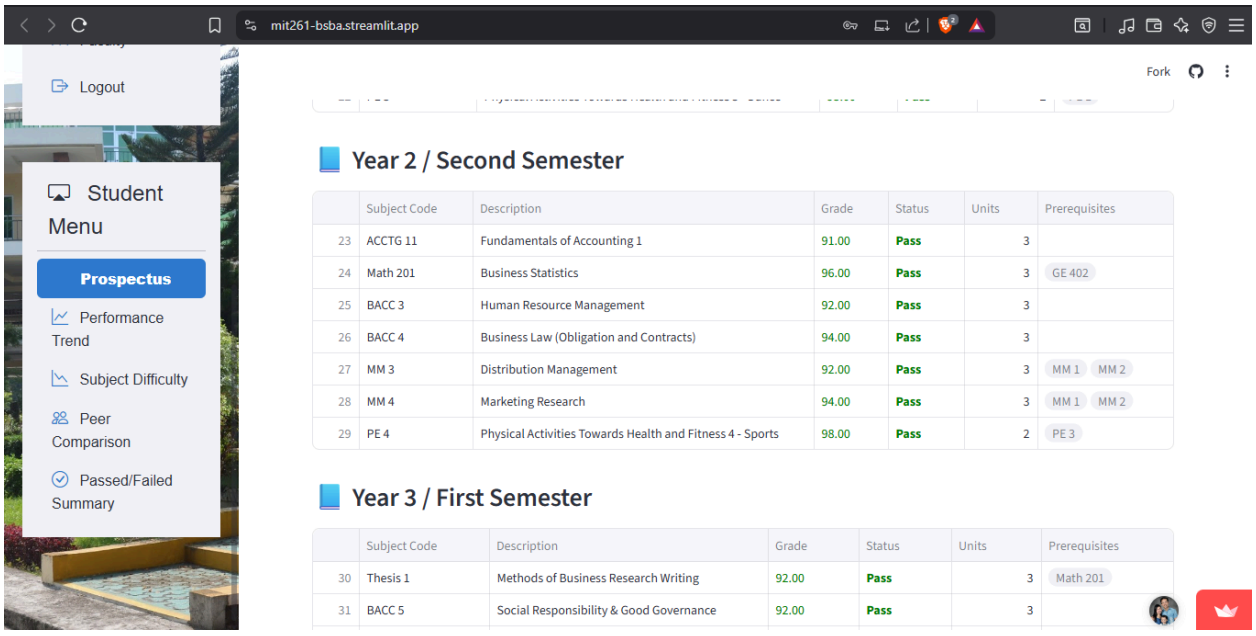
1. Academic Transcript Viewer





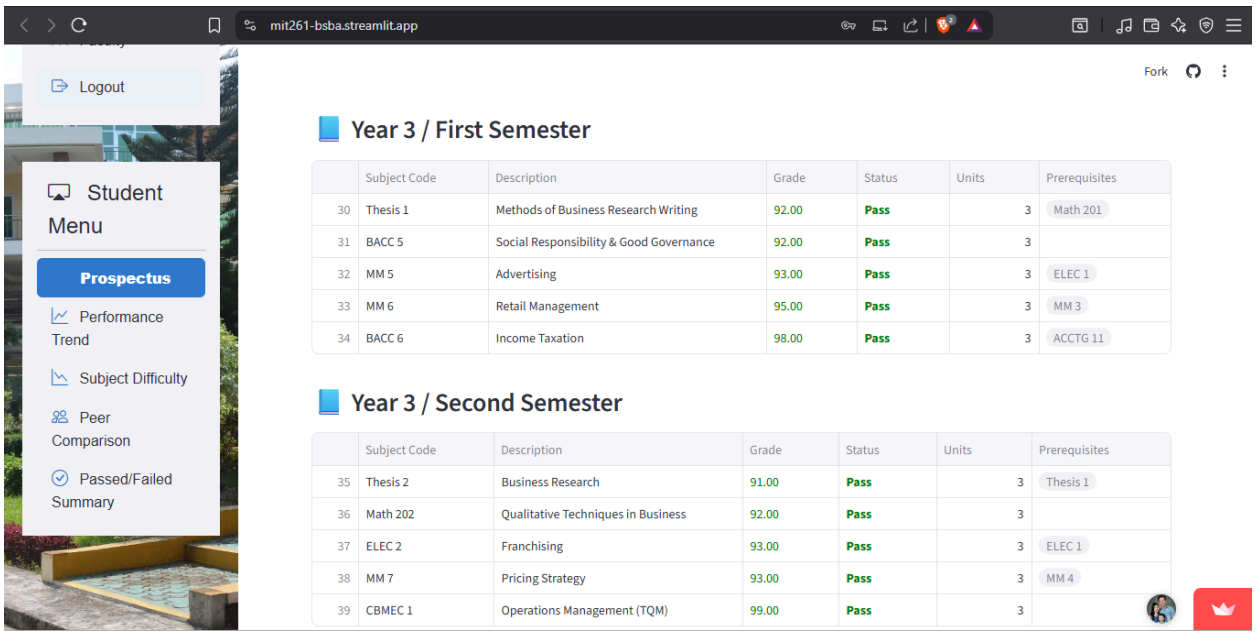

mit261-bsba.streamlit.app
Fork
Logout
Student Menu
Prospectus
Performance Trend
Subject Difficulty
Peer Comparison
Passed/Failed Summary
7 NSTP 1 Civic Welfare Training Services 1 95.00 Pass 3
Year 1 / Second Semester
Subject Code Description Grade Status Units Prerequisites
8 GE 402 Mathematics in the Modern World 96.00 Pass 3
9 GE 302 Ethics 96.00 Pass 3
10 GE 403 Science, Technology and Society 94.00 Pass 3
11 GE 301 Arts Appreciation 96.00 Pass 3
12 GE 501 Living in the IT Era 95.00 Pass 3
13 Mktg 1 Principles of Marketing 94.00 Pass 3
14 PE 2 Physical Activities Towards Health and Fitnes 95.00 Pass 2 PE 1
15 NSTP 2 Civic Welfare Training Services 2 92.00 Pass 3 NSTP 1
Year 2 / First Semester
Subject Code Description Grade Status Units Prerequisites


mit261-bsba.streamlit.app
Fork
Logout
Student Menu
Prospectus
Performance Trend
Subject Difficulty
Peer Comparison
Passed/Failed Summary
15 NSTP 2 Civic Welfare Training Services 2 92.00 Pass 3 NSTP 1
Year 2 / First Semester
Subject Code Description Grade Status Units Prerequisites
16 GE 104 Purposive Communication 90.00 Pass 3
17 GE 102 The Contemporary World 90.00 Pass 3
18 MM 1 Marketing Management 92.00 Pass 3 Mktg 1
19 MM 2 Product Management 91.00 Pass 3 Mktg 1
20 BACC 2 International Business & Trade 98.00 Pass 3 Mktg 1
21 ELEC 1 E-Commerce & Internet Marketing 92.00 Pass 3 Mktg 1
22 PE 3 Physical Activities Towards Health and Fitness 3 - Dance 95.00 Pass 2 PE 1
Year 2 / Second Semester
Subject Code Description Grade Status Units Prerequisites
23 ACCTG 11 Fundamentals of Accounting 1 91.00 Pass 3



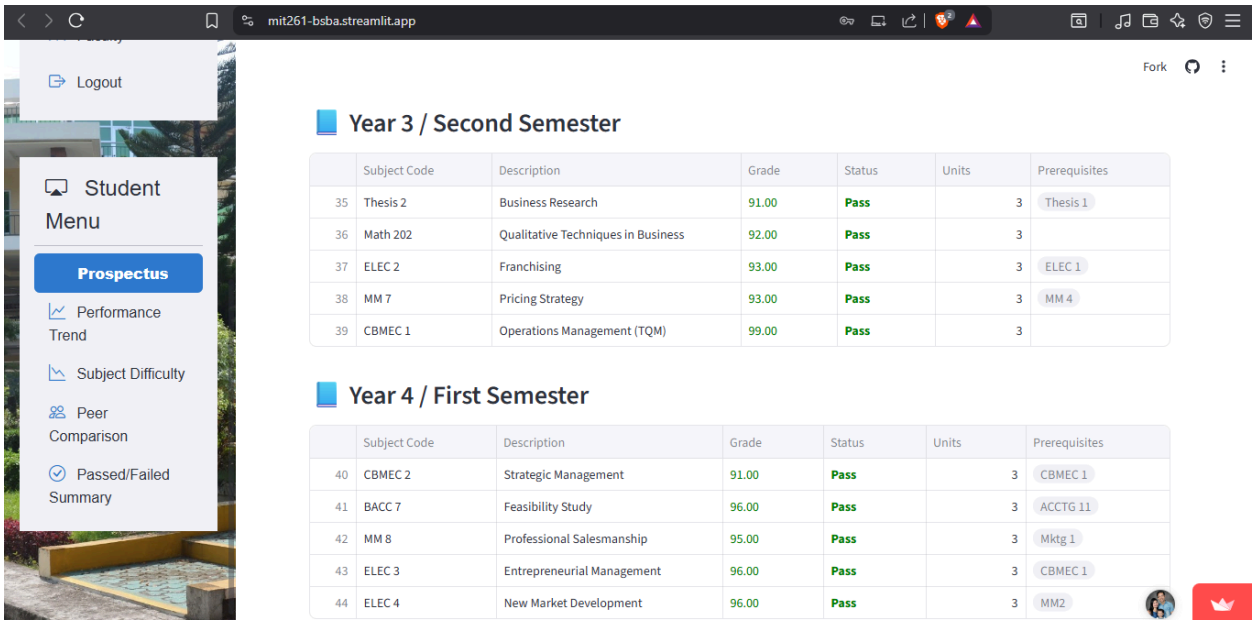
	Subject Code	Description	Grade	Status	Units	Prerequisites
23	ACCTG 11	Fundamentals of Accounting 1	91.00	Pass	3	
24	Math 201	Business Statistics	96.00	Pass	3	GE 402
25	BACC 3	Human Resource Management	92.00	Pass	3	
26	BACC 4	Business Law (Obligation and Contracts)	94.00	Pass	3	
27	MM 3	Distribution Management	92.00	Pass	3	MM 1 MM 2
28	MM 4	Marketing Research	94.00	Pass	3	MM 1 MM 2
29	PE 4	Physical Activities Towards Health and Fitness 4 - Sports	98.00	Pass	2	PE 3

	Subject Code	Description	Grade	Status	Units	Prerequisites
30	Thesis 1	Methods of Business Research Writing	92.00	Pass	3	Math 201
31	BACC 5	Social Responsibility & Good Governance	92.00	Pass	3	



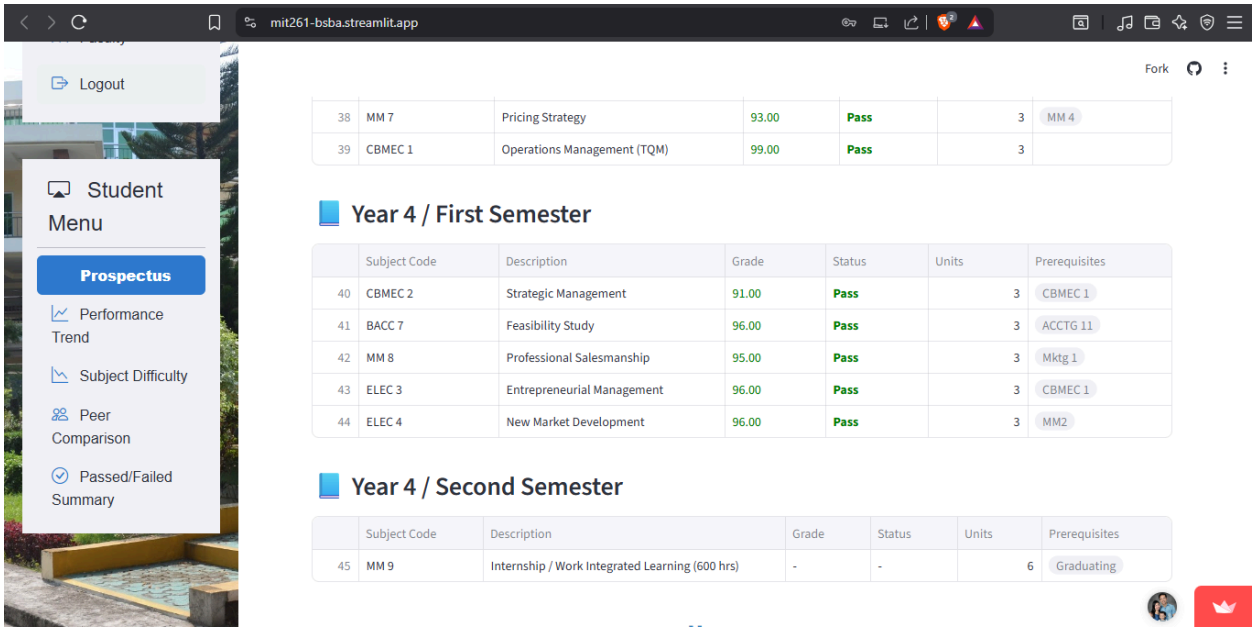
	Subject Code	Description	Grade	Status	Units	Prerequisites
30	Thesis 1	Methods of Business Research Writing	92.00	Pass	3	Math 201
31	BACC 5	Social Responsibility & Good Governance	92.00	Pass	3	
32	MM 5	Advertising	93.00	Pass	3	ELEC 1
33	MM 6	Retail Management	95.00	Pass	3	MM 3
34	BACC 6	Income Taxation	98.00	Pass	3	ACCTG 11

	Subject Code	Description	Grade	Status	Units	Prerequisites
35	Thesis 2	Business Research	91.00	Pass	3	Thesis 1
36	Math 202	Qualitative Techniques in Business	92.00	Pass	3	
37	ELEC 2	Franchising	93.00	Pass	3	ELEC 1
38	MM 7	Pricing Strategy	93.00	Pass	3	MM 4
39	CBMEC 1	Operations Management (TQM)	99.00	Pass	3	



	Subject Code	Description	Grade	Status	Units	Prerequisites
35	Thesis 2	Business Research	91.00	Pass	3	Thesis 1
36	Math 202	Qualitative Techniques in Business	92.00	Pass	3	
37	ELEC 2	Franchising	93.00	Pass	3	ELEC 1
38	MM 7	Pricing Strategy	93.00	Pass	3	MM 4
39	CBMEC 1	Operations Management (TQM)	99.00	Pass	3	

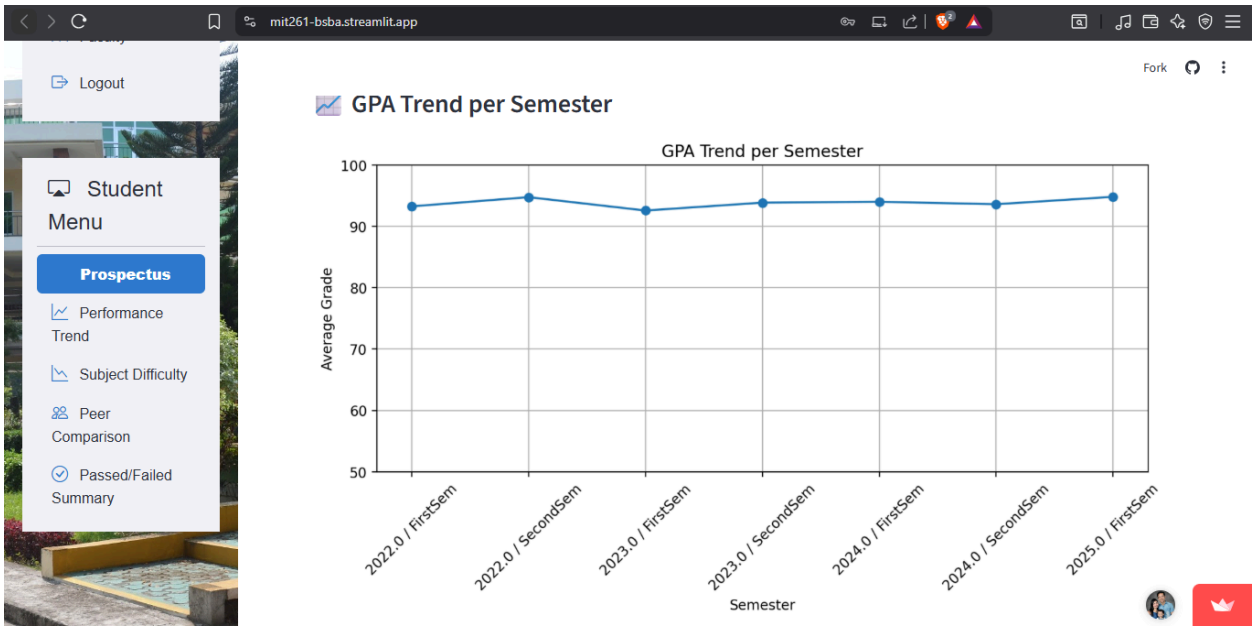
	Subject Code	Description	Grade	Status	Units	Prerequisites
40	CBMEC 2	Strategic Management	91.00	Pass	3	CBMEC 1
41	BACC 7	Feasibility Study	96.00	Pass	3	ACCTG 11
42	MM 8	Professional Salesmanship	95.00	Pass	3	Mktg 1
43	ELEC 3	Entrepreneurial Management	96.00	Pass	3	CBMEC 1
44	ELEC 4	New Market Development	96.00	Pass	3	MM2



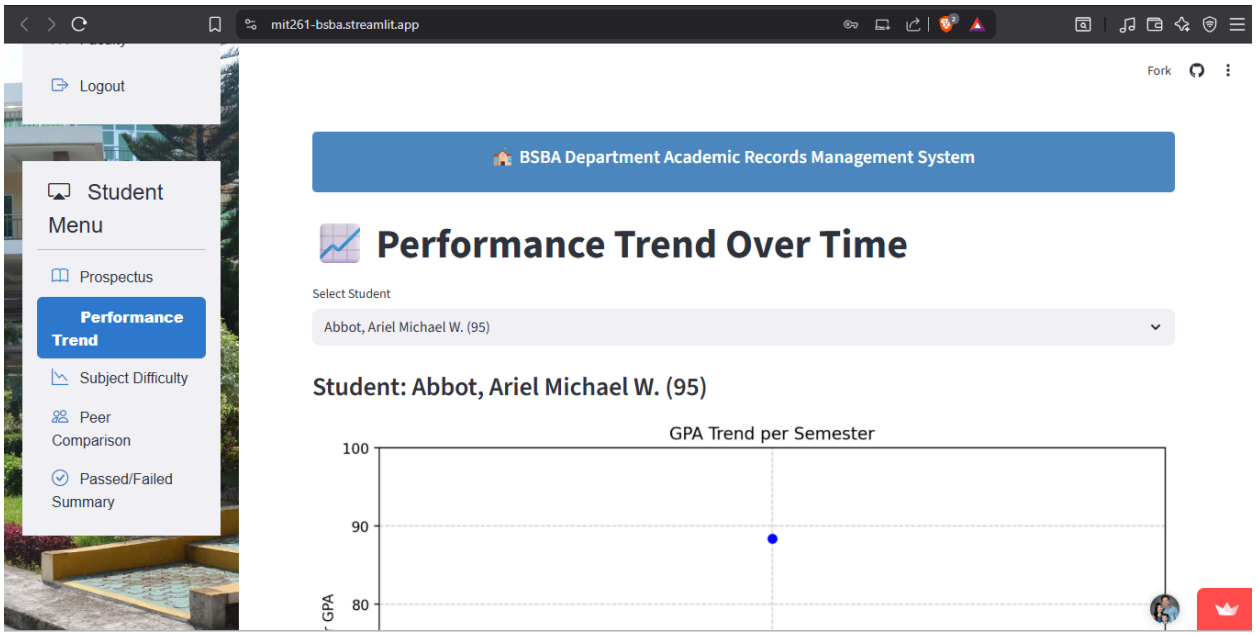
	Subject Code	Description	Grade	Status	Units	Prerequisites
38	MM 7	Pricing Strategy	93.00	Pass	3	MM 4
39	CBMEC 1	Operations Management (TQM)	99.00	Pass	3	

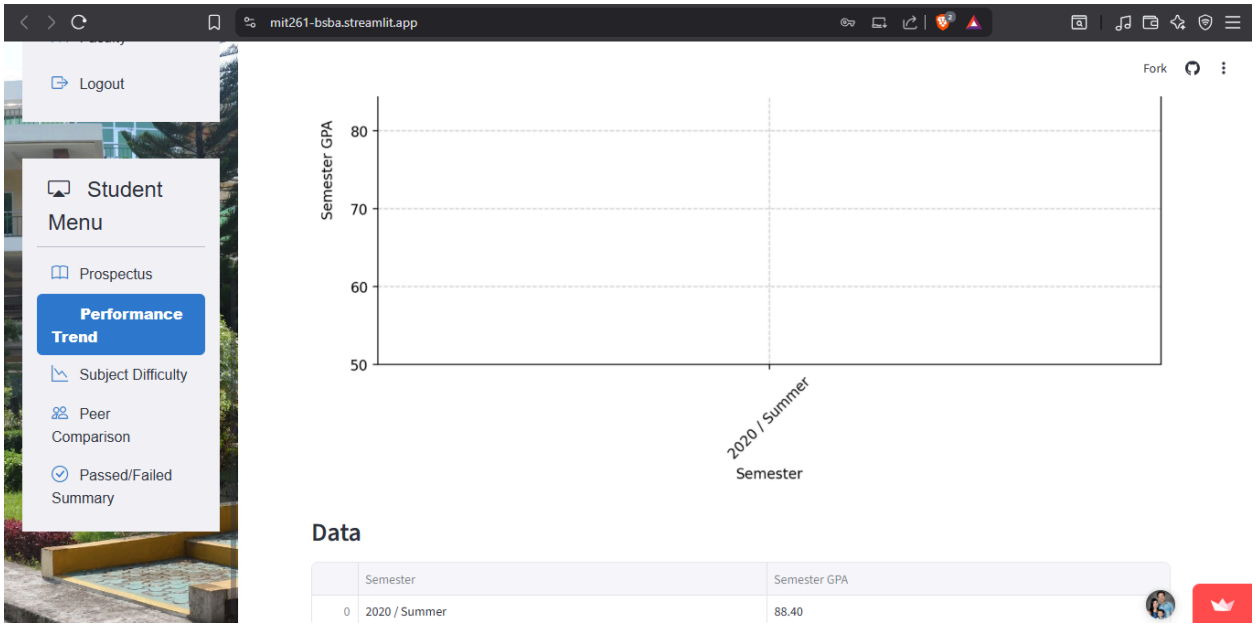
	Subject Code	Description	Grade	Status	Units	Prerequisites
40	CBMEC 2	Strategic Management	91.00	Pass	3	CBMEC 1
41	BACC 7	Feasibility Study	96.00	Pass	3	ACCTG 11
42	MM 8	Professional Salesmanship	95.00	Pass	3	Mktg 1
43	ELEC 3	Entrepreneurial Management	96.00	Pass	3	CBMEC 1
44	ELEC 4	New Market Development	96.00	Pass	3	MM2

	Subject Code	Description	Grade	Status	Units	Prerequisites
45	MM 9	Internship / Work Integrated Learning (600 hrs)	-	-	6	Graduating



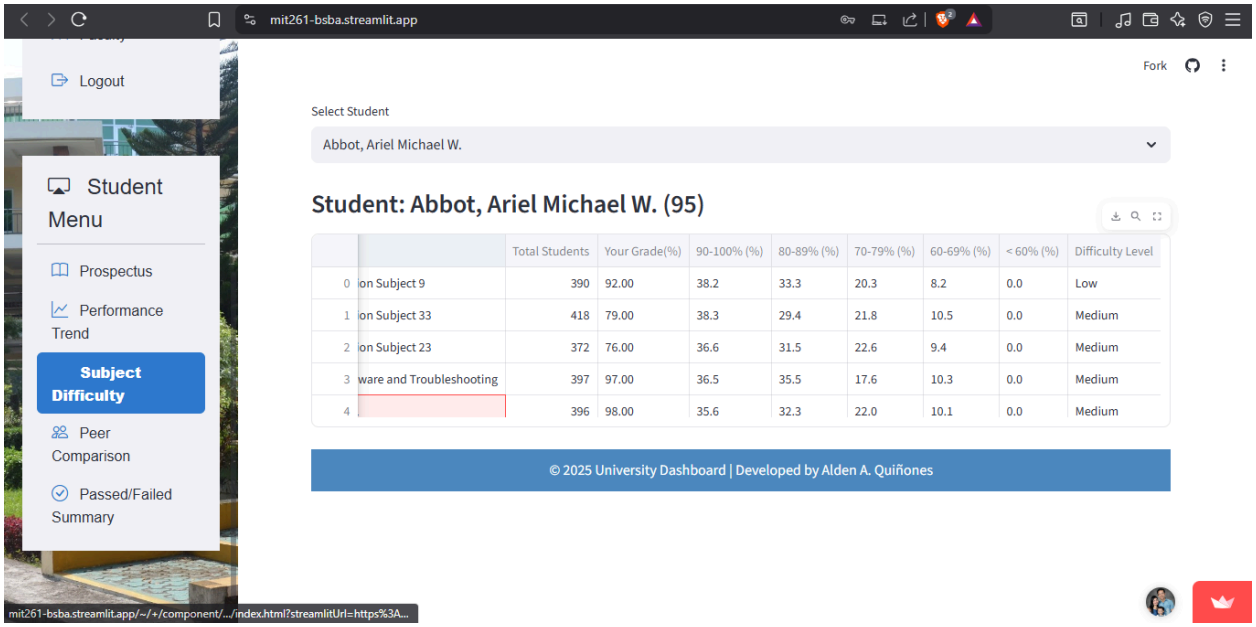
2. Performance Trend Over Time



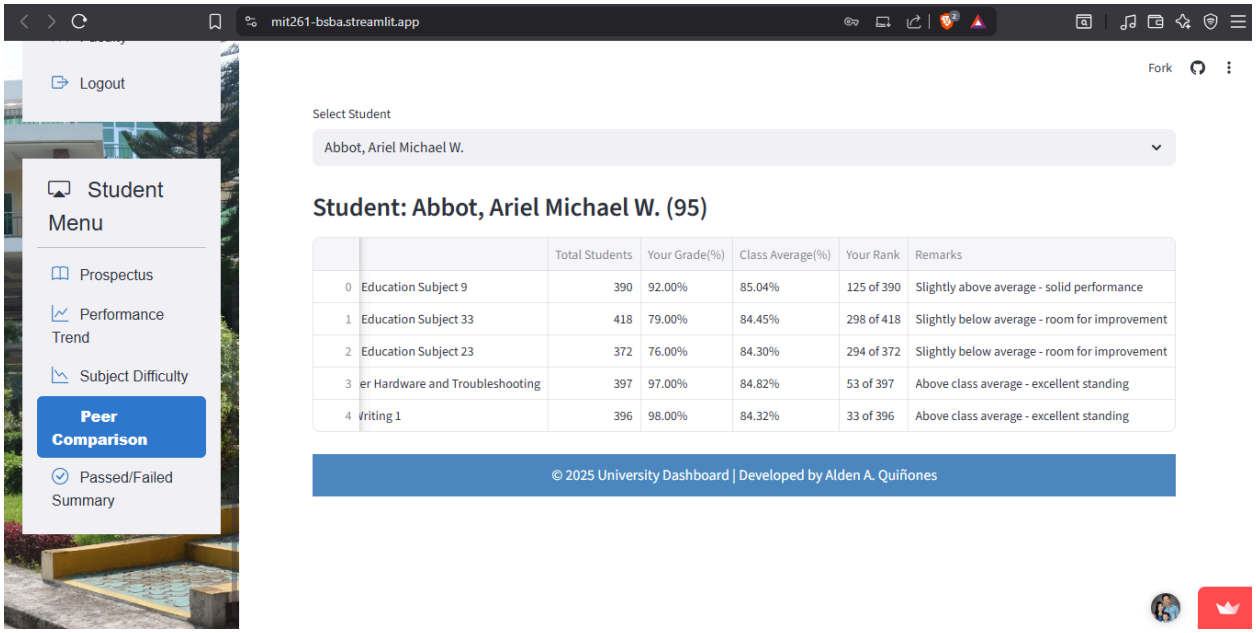


Description: Represents GPA progression across semesters, ideal for a line chart visual.

3. Subject Difficulty Ratings

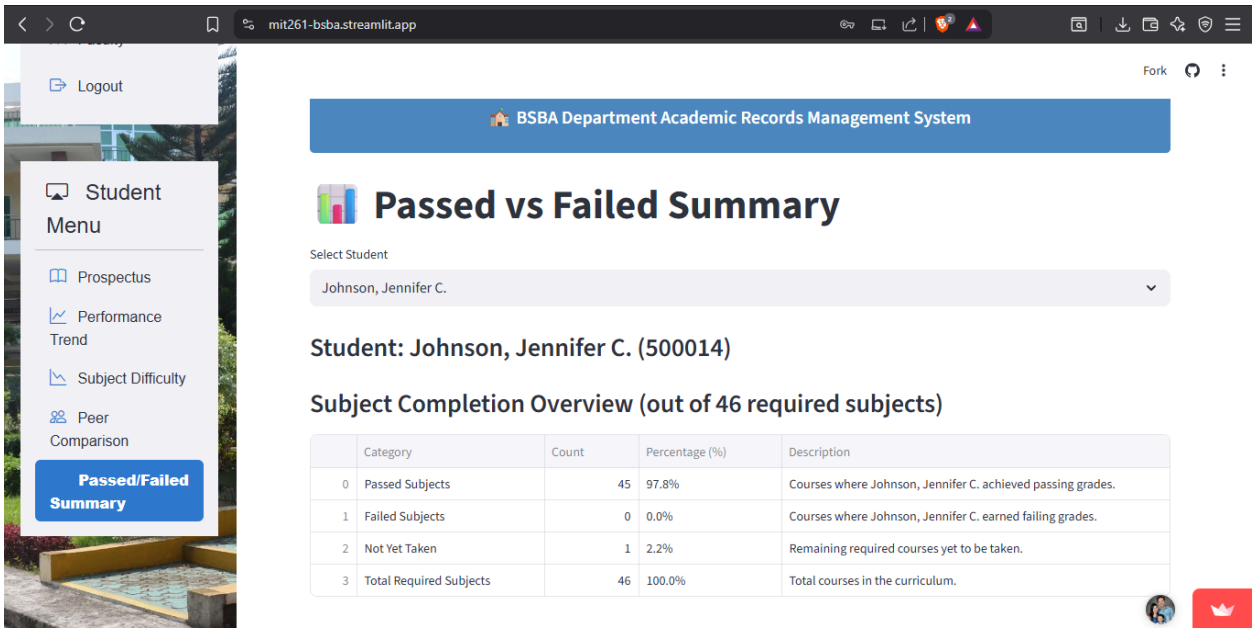


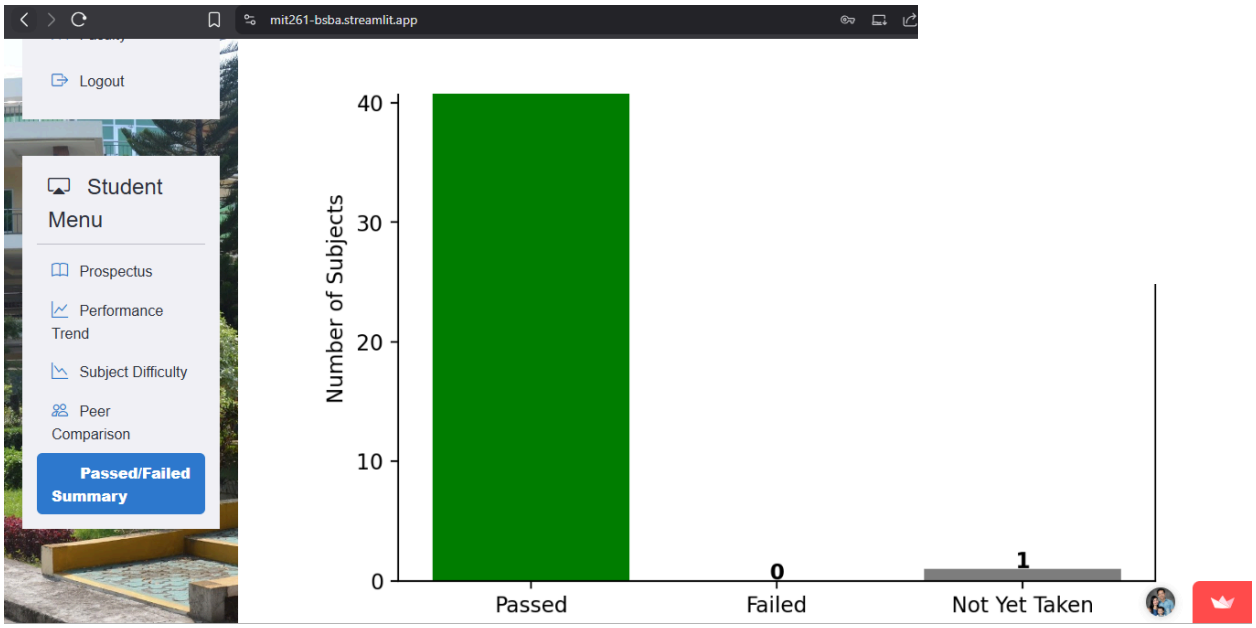
4. Comparison with Class Average



Description: Highlights how the student's performance stacks up against peers.

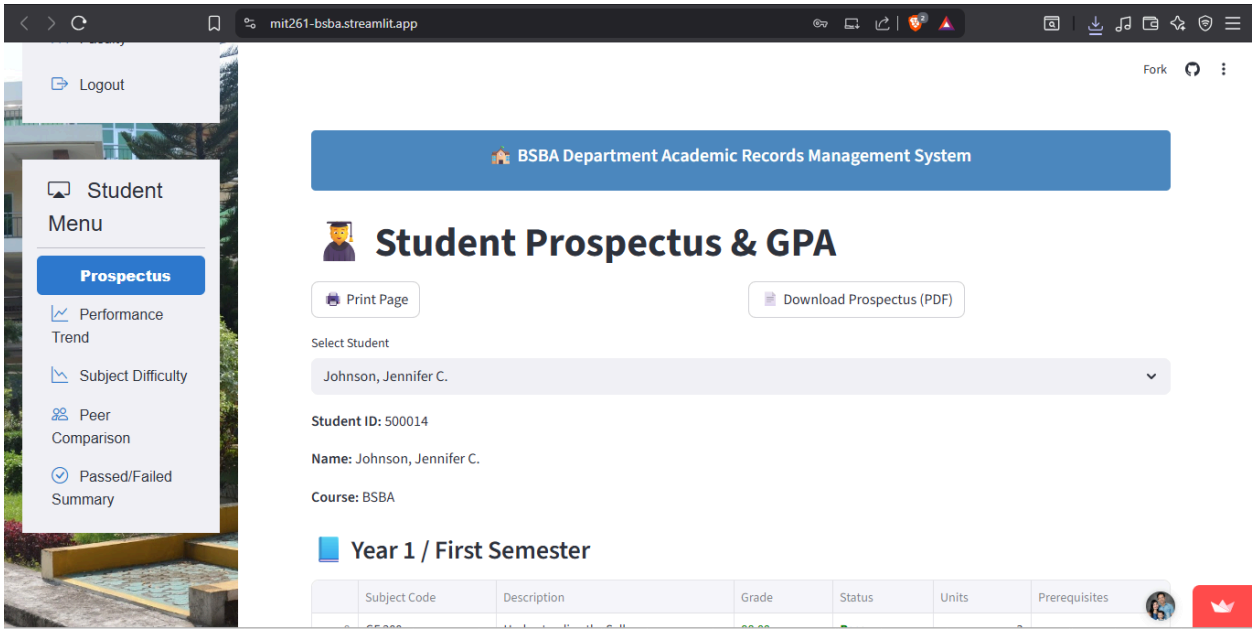
5. Passed vs Failed Summary

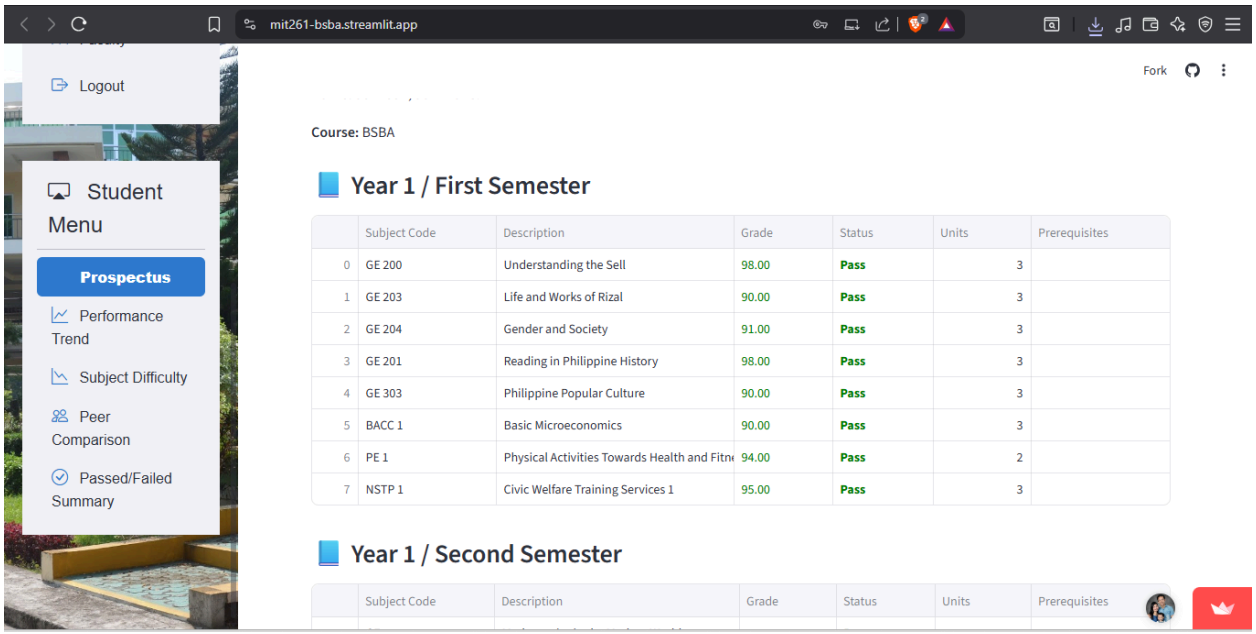


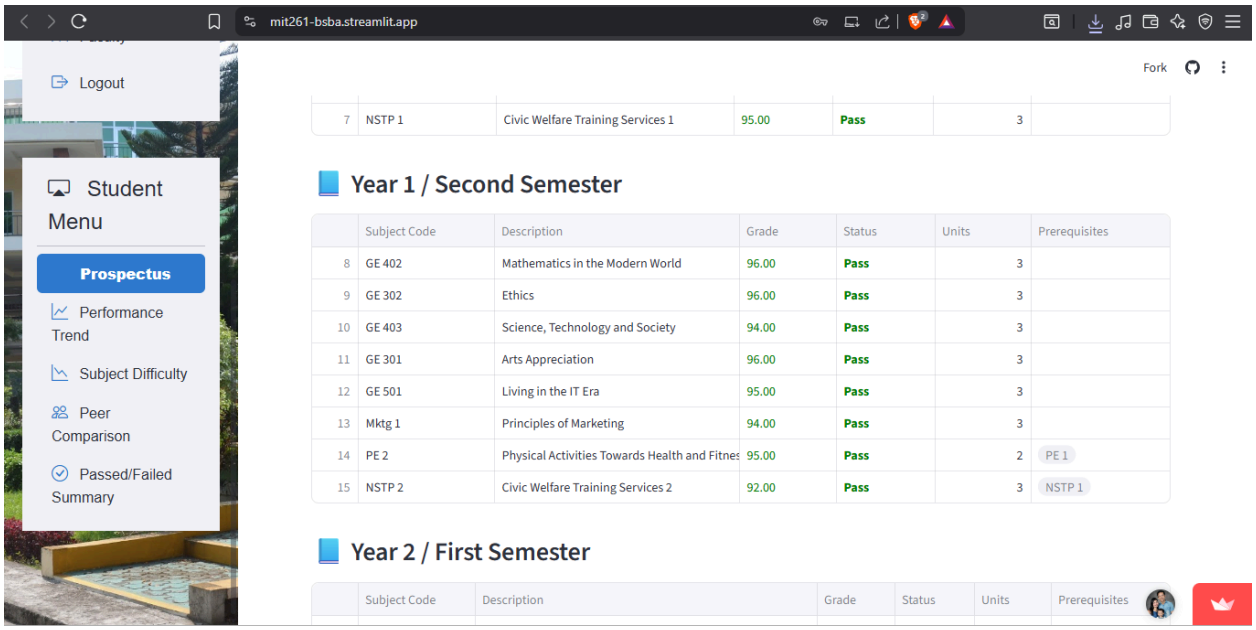


Description: A simple summary of academic outcomes—ideal for pie or bar chart depiction.

6.Curriculum and Subject Viewer







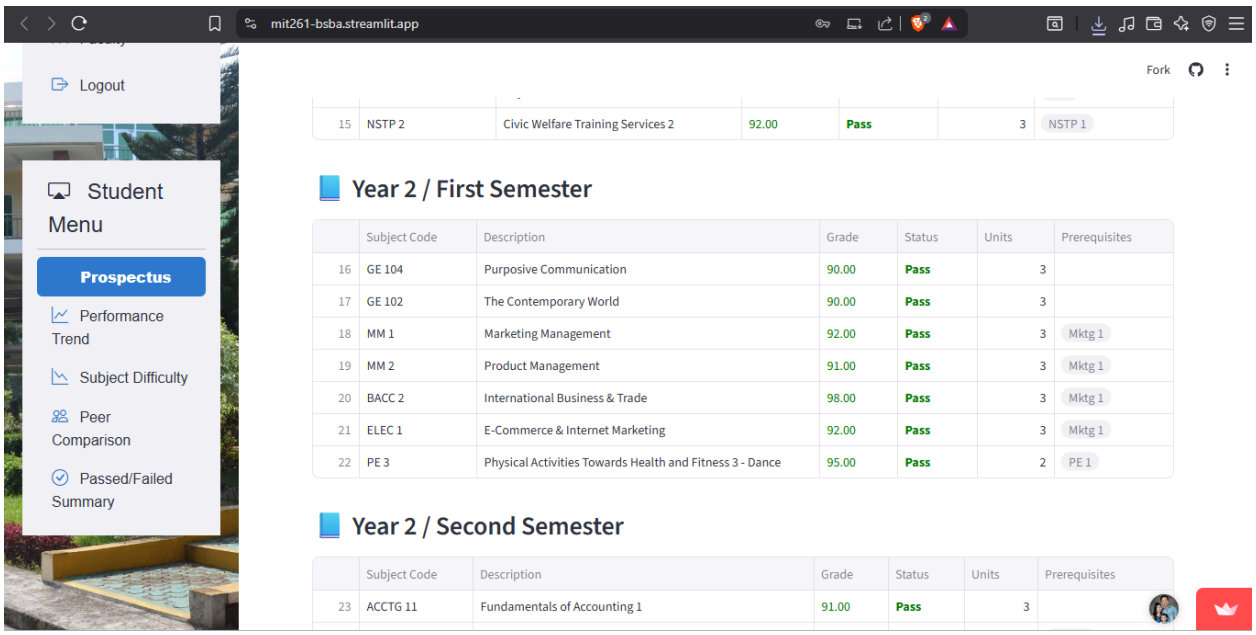
7	NSTP 1	Civic Welfare Training Services 1	95.00	Pass	3	

Year 1 / Second Semester

	Subject Code	Description	Grade	Status	Units	Prerequisites
8	GE 402	Mathematics in the Modern World	96.00	Pass	3	
9	GE 302	Ethics	96.00	Pass	3	
10	GE 403	Science, Technology and Society	94.00	Pass	3	
11	GE 301	Arts Appreciation	96.00	Pass	3	
12	GE 501	Living in the IT Era	95.00	Pass	3	
13	Mktg 1	Principles of Marketing	94.00	Pass	3	
14	PE 2	Physical Activities Towards Health and Fitnes	95.00	Pass	2	PE 1
15	NSTP 2	Civic Welfare Training Services 2	92.00	Pass	3	NSTP 1

Year 2 / First Semester

	Subject Code	Description	Grade	Status	Units	Prerequisites



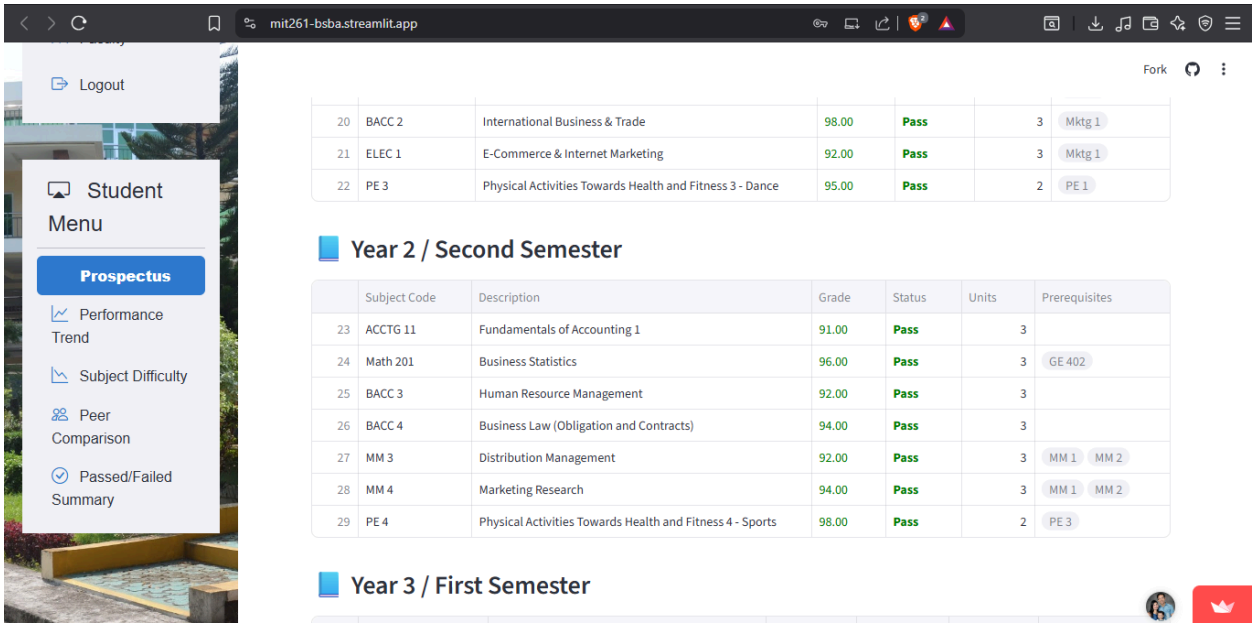
15	NSTP 2	Civic Welfare Training Services 2	92.00	Pass	3	NSTP 1

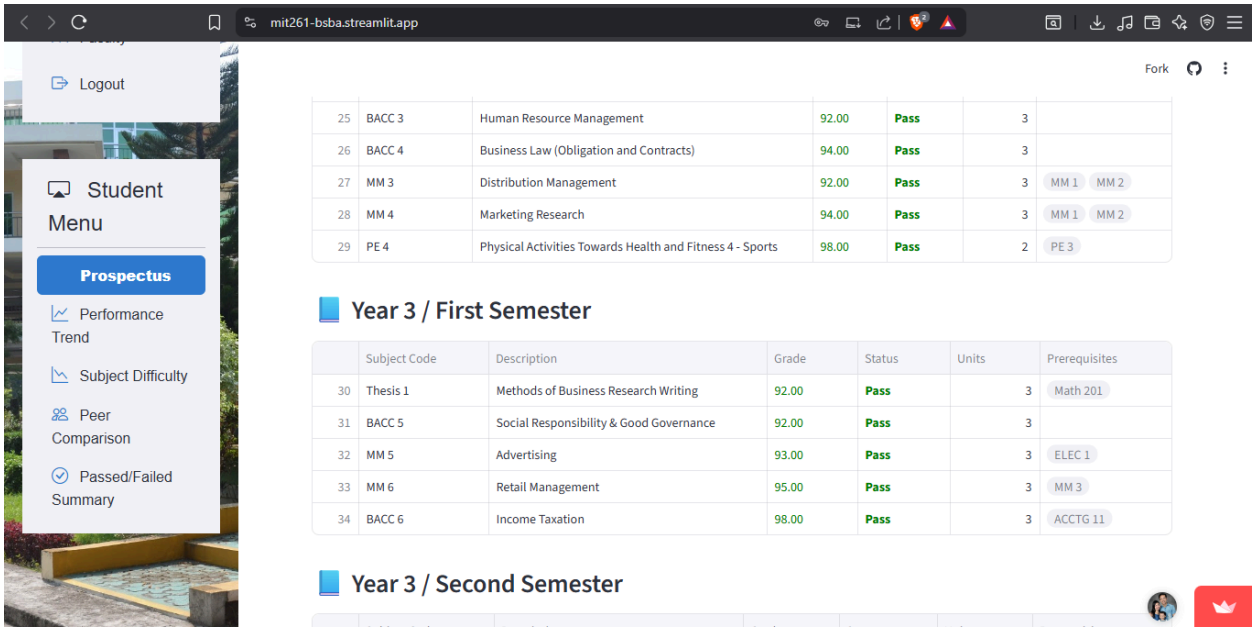
Year 2 / First Semester

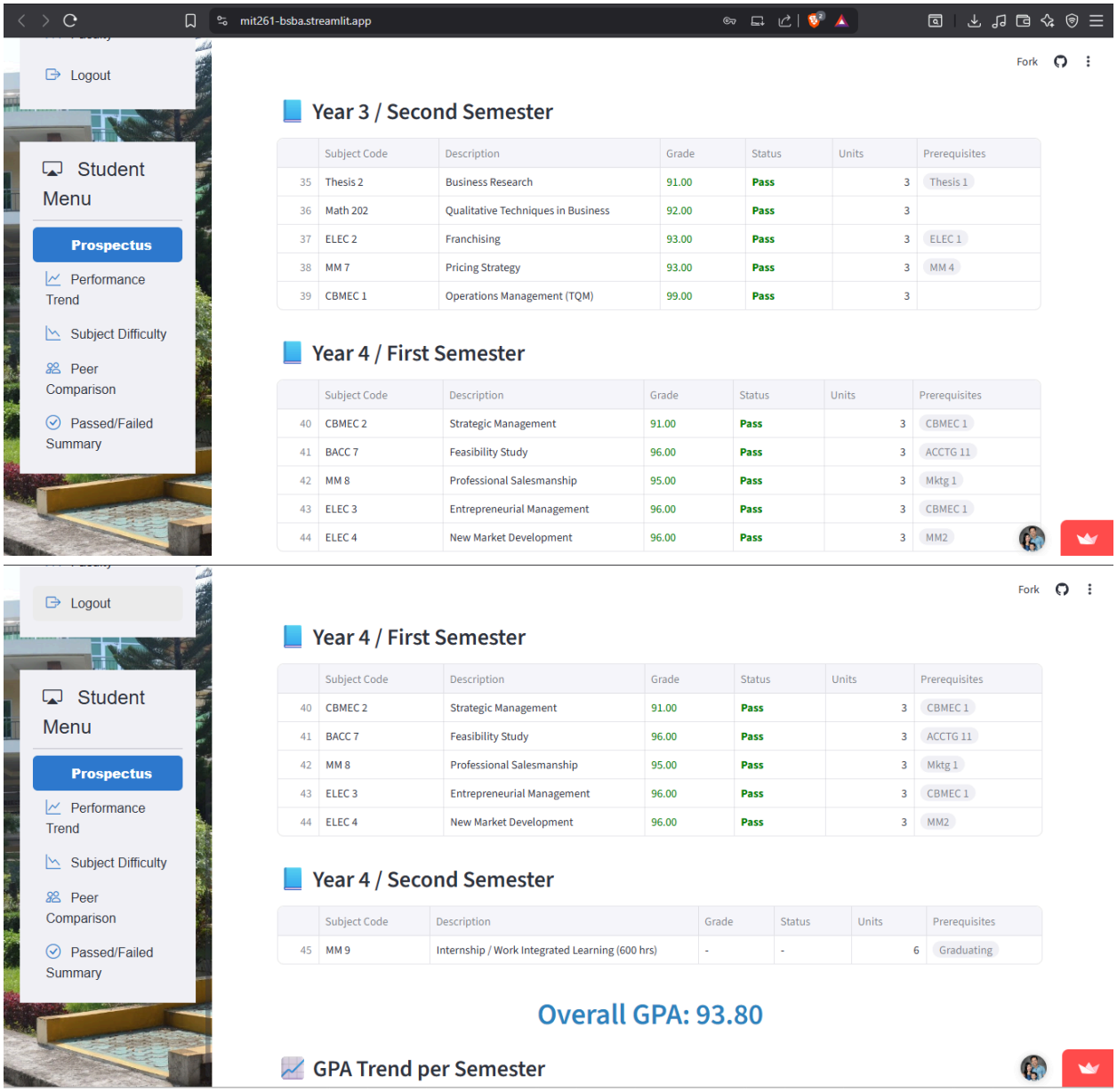
	Subject Code	Description	Grade	Status	Units	Prerequisites
16	GE 104	Purposive Communication	90.00	Pass	3	
17	GE 102	The Contemporary World	90.00	Pass	3	
18	MM 1	Marketing Management	92.00	Pass	3	Mktg 1
19	MM 2	Product Management	91.00	Pass	3	Mktg 1
20	BACC 2	International Business & Trade	98.00	Pass	3	Mktg 1
21	ELEC 1	E-Commerce & Internet Marketing	92.00	Pass	3	Mktg 1
22	PE 3	Physical Activities Towards Health and Fitness 3 - Dance	95.00	Pass	2	PE 1

Year 2 / Second Semester

	Subject Code	Description	Grade	Status	Units	Prerequisites
23	ACCTG 11	Fundamentals of Accounting 1	91.00	Pass	3	


mit261-bsba.streamlit.app
Fork
Logout
Student Menu
Prospectus
Performance Trend
Subject Difficulty
Peer Comparison
Passed/Failed Summary
20 BACC 2 International Business & Trade 98.00 Pass 3 Mktg 1
21 ELEC 1 E-Commerce & Internet Marketing 92.00 Pass 3 Mktg 1
22 PE 3 Physical Activities Towards Health and Fitness 3 - Dance 95.00 Pass 2 PE 1
Year 2 / Second Semester
23 ACCTG 11 Fundamentals of Accounting 1 91.00 Pass 3
24 Math 201 Business Statistics 96.00 Pass 3 GE 402
25 BACC 3 Human Resource Management 92.00 Pass 3
26 BACC 4 Business Law (Obligation and Contracts) 94.00 Pass 3
27 MM 3 Distribution Management 92.00 Pass 3 MM 1 MM 2
28 MM 4 Marketing Research 94.00 Pass 3 MM 1 MM 2
29 PE 4 Physical Activities Towards Health and Fitness 4 - Sports 98.00 Pass 2 PE 3
Year 3 / First Semester


mit261-bsba.streamlit.app
Fork
Logout
Student Menu
Prospectus
Performance Trend
Subject Difficulty
Peer Comparison
Passed/Failed Summary
25 BACC 3 Human Resource Management 92.00 Pass 3
26 BACC 4 Business Law (Obligation and Contracts) 94.00 Pass 3
27 MM 3 Distribution Management 92.00 Pass 3 MM 1 MM 2
28 MM 4 Marketing Research 94.00 Pass 3 MM 1 MM 2
29 PE 4 Physical Activities Towards Health and Fitness 4 - Sports 98.00 Pass 2 PE 3
Year 3 / First Semester
30 Thesis 1 Methods of Business Research Writing 92.00 Pass 3 Math 201
31 BACC 5 Social Responsibility & Good Governance 92.00 Pass 3
32 MM 5 Advertising 93.00 Pass 3 ELEC 1
33 MM 6 Retail Management 95.00 Pass 3 MM 3
34 BACC 6 Income Taxation 98.00 Pass 3 ACCTG 11
Year 3 / Second Semester

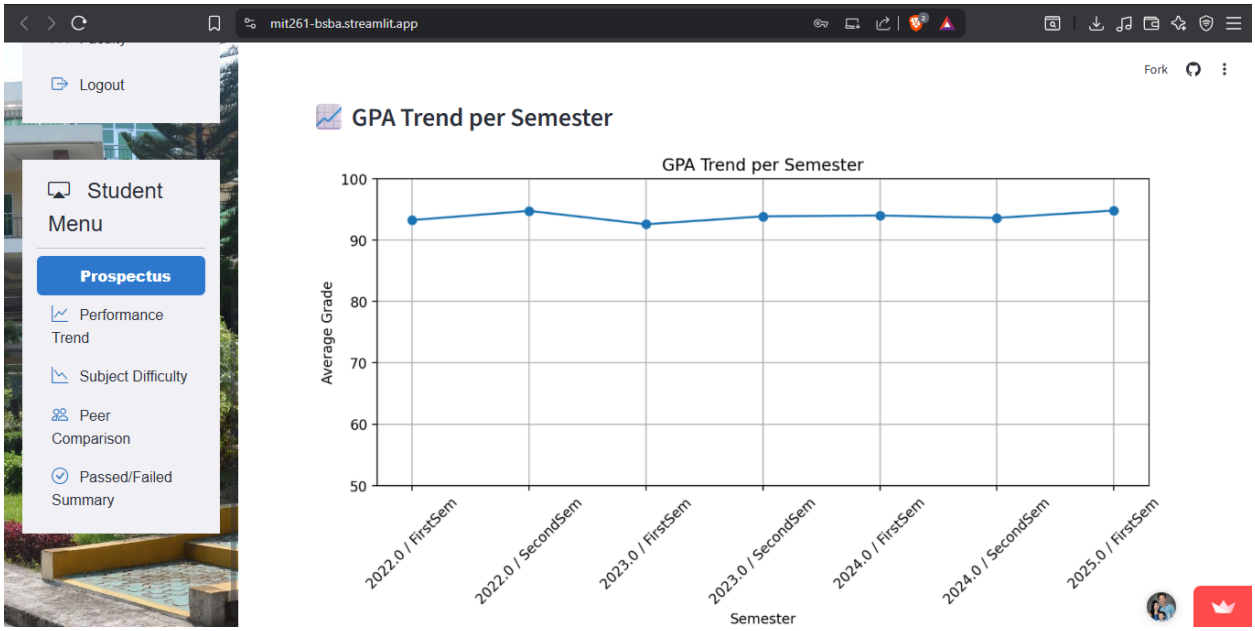


	Subject Code	Description	Grade	Status	Units	Prerequisites
35	Thesis 2	Business Research	91.00	Pass	3	Thesis 1
36	Math 202	Qualitative Techniques in Business	92.00	Pass	3	
37	ELEC 2	Franchising	93.00	Pass	3	ELEC 1
38	MM 7	Pricing Strategy	93.00	Pass	3	MM 4
39	CBMEC 1	Operations Management (TQM)	99.00	Pass	3	

	Subject Code	Description	Grade	Status	Units	Prerequisites
40	CBMEC 2	Strategic Management	91.00	Pass	3	CBMEC 1
41	BACC 7	Feasibility Study	96.00	Pass	3	ACCTG 11
42	MM 8	Professional Salesmanship	95.00	Pass	3	Mktg 1
43	ELEC 3	Entrepreneurial Management	96.00	Pass	3	CBMEC 1
44	ELEC 4	New Market Development	96.00	Pass	3	MM2

	Subject Code	Description	Grade	Status	Units	Prerequisites
40	CBMEC 2	Strategic Management	91.00	Pass	3	CBMEC 1
41	BACC 7	Feasibility Study	96.00	Pass	3	ACCTG 11
42	MM 8	Professional Salesmanship	95.00	Pass	3	Mktg 1
43	ELEC 3	Entrepreneurial Management	96.00	Pass	3	CBMEC 1
44	ELEC 4	New Market Development	96.00	Pass	3	MM2

	Subject Code	Description	Grade	Status	Units	Prerequisites
45	MM 9	Internship / Work Integrated Learning (600 hrs)	-	-	6	Graduating


mit261-bsba.streamlit.app
Logout
Student Menu
Prospectus
Performance Trend
Subject Difficulty
Peer Comparison
Passed/Failed Summary
Fork
GPA Trend per Semester
GPA Trend per Semester
Average Grade
100
90
80
70
60
50
2022.0 / FirstSem
2022.0 / SecondSem
2023.0 / FirstSem
2023.0 / SecondSem
2024.0 / FirstSem
2024.0 / SecondSem
2025.0 / FirstSem
Semester

Source code for each dashboard and reports

Registrar's Office Dashboard Reports

1. Student Academic Standing Report

a. Dean's List (Top 10 Students)

```
# ------------------------------
# A. Dean's List
# ------------------------------
st.markdown("### A. Dean's List (:rainbow[Top 10 Students])")
st.markdown("**Criteria:** No grade < 85% & GPA >= 90%")

with st.spinner(f"Preparing data for student academic Stand Reporting - Dean's List.",
show_time=True):
   df_deans = r.get_deans_list(course=course_filter, year_level=year_level_filter)  # fetch
data

option_deans = None  # ensure variable always exists

if df_deans.empty:
   st.info("No Dean's List entries found for the chosen semester/year. "
         "This means no students met the GPA and grade requirements with the applied
filters.")
else:
   # Display table
   st.dataframe(df_deans, width="stretch")

   # Chart data
   names_deans = df_deans["Name"].tolist()
   gpas_deans = df_deans["GPA"].tolist()

   option_deans = {
      "tooltip": {"trigger": "axis", "axisPointer": {"type": "shadow"}},
      "xAxis": {"type": "value", "name": "GPA", "min": 90},
      "yAxis": {"type": "category", "data": names_deans[::-1]},
      "series": [
         {
            "name": "GPA",
            "type": "bar",
```

```
            "data": gpas_deans[::-1],
            "label": {"show": True, "position": "right", "formatter": "{c}"},
            "itemStyle": {
              "color": {
                "type": "linear",
                "x": 0, "y": 0, "x2": 1, "y2": 0,
                "colorStops": [
                  {"offset": 0, "color": "#3b82f6"},
                  {"offset": 1, "color": "#10b981"},
                ],
              }
            },
          }
        ],
      }
```

b. Academic Probation

```
# ------------------------------
# B. Academic Probation
# ------------------------------
st.markdown("### B. Academic Probation (10 Students)")
st.markdown("**Criteria:** No grade < 75 OR >= 30% FAILS")
with st.spinner(f"Preparing data for academic probation.", show_time=True):
    df_probation = r.get_academic_probation_batch_checkpoint(course=course_filter,
year_level=year_level_filter)
option_prob = None  # ensure variable always exists
if df_probation.empty:
    st.info("No students are under academic probation with the applied filters.")
else:
    # Display table
    st.dataframe(df_probation, width="stretch")
    # Chart data (safe conversion)
    required_cols = ["Name", "GPA", "Fail%"]
    if not all(col in df_probation.columns for col in required_cols):
 st.error(f"❌ Missing required columns: {required_cols}")
 st.write("Available columns:", df_probation.columns.tolist())
    else:
 df_probation["GPA"] = pd.to_numeric(df_probation["GPA"], errors="coerce")
```

```
df_probation["Fail%"] = pd.to_numeric(df_probation["Fail%"], errors="coerce")

names_prob = df_probation["Name"].astype(str).tolist()
gpas_prob = df_probation["GPA"].fillna(0).tolist()
fails_prob = df_probation["Fail%"].fillna(0).tolist()

option_prob = {
    "tooltip": {"trigger": "axis"},
    "legend": {"data": ["GPA", "Fail%"]},
    "xAxis": [{"type": "value", "name": "Score"}],
    "yAxis": [{"type": "category", "data": names_prob[::-1]}],
    "series": [
  {
"name": "GPA",
"type": "bar",
"data": gpas_prob[::-1],
"label": {"show": True, "position": "right"},
"itemStyle": {"color": "#ef4444"},
  },
  {
"name": "Fail%",
"type": "line",
"data": fails_prob[::-1],
"label": {"show": True, "position": "top"},
"lineStyle": {"color": "#f59e0b", "width": 2},
"symbol": "circle",
"symbolSize": 8,
  },
    ],
 }
```

2. Subject Pass/Fail Distribution

```
# ------------------------------
# 2. Subject Pass/Fail Distribution
# ------------------------------
elif report == "2. Subject Pass/Fail Distribution":
   st.subheader("📊 Subject Pass/Fail Distribution")
```

```
    # --- Filters ---
    courses =  r2.get_courses() + ["All"]
    year_levels =  r2.get_year_levels() + ["All"]

    col1, col2 = st.columns(2)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)

    # --- Apply filters ---
    course_filter = selected_course if selected_course != "All" else None
    year_level_filter = selected_year_level if selected_year_level != "All" else None

    with st.spinner(f"Preparing data for {report}.", show_time = True):
        df_subjects = r.get_subject_pass_fail(course=course_filter, year_level=year_level_filter)  #
columns: ['Subject Code', 'Subject Name', 'Semester', 'Pass Count', 'Fail Count', 'Pass %', 'Fail %']
    st.dataframe(df_subjects)

    st.markdown("""
    **Insight:**
    Provides a clear picture of **subject-level performance per semester**, enabling targeted
support for subjects
    with higher failure rates. Administrators can plan remediation programs effectively.
    """)

    if not df_subjects.empty:
        # Aggregate data by Subject Code and Subject Name
        df_chart = df_subjects.groupby(['Subject Code', 'Subject Name']).agg(
            {'Pass Count': 'sum', 'Fail Count': 'sum'}
        ).reset_index()

        # Calculate totals and percentages (good practice)
        df_chart['Total Count'] = df_chart['Pass Count'] + df_chart['Fail Count']
        df_chart['Pass %'] = (df_chart['Pass Count'] / df_chart['Total Count']) * 100
        df_chart['Fail %'] = (df_chart['Fail Count'] / df_chart['Total Count']) * 100

        # Stacked ECharts
        option = {
            "tooltip": {"trigger": "axis", "axisPointer": {"type": "shadow"}},
            "legend": {"data": ["Pass Count", "Fail Count"]},  # optional order for legend
```

```
        "xAxis": [
          {
            "type": "category",
            "data": df_chart["Subject Name"].tolist(),
            "axisLabel": {"interval": 0, "rotate": 30},
          }
        ],
        "yAxis": [{"type": "value", "name": "Students"}],
        "series": [
          {
            "name": "Fail Count",
            "type": "bar",
            "stack": "total",
            "emphasis": {"focus": "series"},
            "data": df_chart["Fail Count"].tolist(),
            "itemStyle": {"color": "#ef4444"},
          },
          {
            "name": "Pass Count",
            "type": "bar",
            "stack": "total",
            "emphasis": {"focus": "series"},
            "data": df_chart["Pass Count"].tolist(),
            "itemStyle": {"color": "#3b82f6"},
          }
        ],
      }
      st_echarts(options=option, height="500px")
```

3. Enrollment Trend Analysis

```
# ------------------------------
# 3. Enrollment Trend Analysis
# ------------------------------
elif report == "3. Enrollment Trend Analysis":
    st.subheader("📈 Enrollment Trend Analysis")

    # --- Filters ---
    courses =  r2.get_courses() +  ["All"]
```

```
    year_levels =  r2.get_year_levels() +  ["All"]

    col1, col2 = st.columns(2)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)

    # --- Apply filters ---
    course_filter = selected_course if selected_course != "All" else None
    year_level_filter = selected_year_level if selected_year_level != "All" else None

    with st.spinner(f"Preparing data for {report}.", show_time = True):
        df_enrollment = r.get_enrollment_trend(course=course_filter, year_level=year_level_filter)  # columns: ['Semester', 'Total Enrollment', 'New Enrollees', 'Dropouts', 'Retention Rate (%)']
    st.dataframe(df_enrollment)

    st.markdown("""
    **Insight:**
    Observe semester-to-semester enrollment trends, **track retention rates**, and identify patterns of student dropouts.
    Helps institutions plan resource allocation and intervention strategies.
    """)

    # Prepare ECharts options
    option = {
        "tooltip": {"trigger": "axis"},
        "legend": {"data": ["Total Enrollment", "New Enrollees", "Dropouts", "Retention Rate (%)"]},
        "xAxis": {"type": "category", "data": df_enrollment["Semester"].tolist()},
        "yAxis": [
            {"type": "value", "name": "Students"},
            {"type": "value", "name": "Retention (%)", "min": 0, "max": 100}
        ],
        "series": [
            {
                "name": "Total Enrollment",
                "type": "bar",
                "data": df_enrollment["Total Enrollment"].tolist()
            },
            {
                "name": "New Enrollees",
```

```
            "type": "bar",
            "data": df_enrollment["New Enrollees"].tolist()
        },
        {
            "name": "Dropouts",
            "type": "bar",
            "data": df_enrollment["Dropouts"].tolist()
        },
        {
            "name": "Retention Rate (%)",
            "type": "line",
            "yAxisIndex": 1,
            "data": df_enrollment["Retention Rate (%)"].tolist()
        }
    ]
}

st_echarts(options=option, height="450px")
```

4. Incomplete Grades Report

```
# ------------------------------
# 4. Incomplete Grades
# ------------------------------
elif report == "4. Incomplete Grades":
    st.subheader("⚠️ Incomplete Grades Report")

    # --- Filters ---
    courses =  r2.get_courses() + ["All"]
    year_levels = r2.get_year_levels() + ["All"]

    col1, col2 = st.columns(2)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)

    # --- Apply filters ---
    course_filter = selected_course if selected_course != "All" else None
```

```
    year_level_filter = selected_year_level if selected_year_level != "All" else None

    with st.spinner(f"Preparing data for {report}.", show_time = True):
        df_incomplete = r.get_incomplete_grades(course=course_filter, year_level=year_level_filter)  # columns: ['Student ID', 'Name', 'Course Code', 'Course Title', 'Term', 'Grade Status']
    st.dataframe(df_incomplete)

    st.markdown("""
    **Insight:**
    Enables the Registrar’s Office to follow up with **students and instructors** regarding incomplete or missing grades,
    ensuring timely grade submissions.
    """)

    if not df_incomplete.empty:
        # --- Chart ---
        st.markdown("### 📊 Incomplete Grades per Course")

        # Group by course and count
        incomplete_counts = df_incomplete.groupby("Course Title").size().reset_index(name="Count")
        incomplete_counts = incomplete_counts.sort_values("Count", ascending=False)

        option = {
            "tooltip": {"trigger": "axis", "axisPointer": {"type": "shadow"}},
            "xAxis": {
                "type": "category",
                "data": incomplete_counts["Course Title"].tolist(),
                "axisLabel": {"interval": 0, "rotate": 30},
            },
            "yAxis": {"type": "value", "name": "Number of Incomplete Grades"},
            "series": [{
                "name": "Incomplete Grades",
                "type": "bar",
                "data": incomplete_counts["Count"].tolist(),
                "itemStyle": {"color": "#f59e0b"},
                "label": {"show": True, "position": "top"}
            }]
        }
        st_echarts(options=option, height="500px")
```

5. Retention and Dropout Rates

```
# ------------------------------
# 5. Retention and Dropout Rates
# ------------------------------
elif report == "5. Retention and Dropout Rates":
    st.subheader("📊 Retention and Dropout Rates")

    # --- Filters ---
    courses = r2.get_courses() + ["All"]
    year_levels = r2.get_year_levels() + ["All"]

    col1, col2 = st.columns(2)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)

    # --- Apply filters ---
    course_filter = selected_course if selected_course != "All" else None
    year_level_filter = selected_year_level if selected_year_level != "All" else None

    with st.spinner(f"Preparing data for {report}.", show_time = True):
        df_retention = r.get_retention_rates(course=course_filter, year_level=year_level_filter)  #
columns: ['Semester to Semester', 'Retained', 'Dropped Out', 'Retention Rate (%)']
        print('df_retention2:',df_retention)
    st.dataframe(df_retention)

    st.markdown("""
    **Insight:**
    Measures **student persistence** across semesters and identifies retention issues early,
    providing actionable data for academic planning and intervention programs.
    """)

    if not df_retention.empty:
        # --- Chart ---
        st.markdown("### 📊 Retention vs. Dropout")

        option = {
            "tooltip": {"trigger": "axis"},
```

```
    "legend": {"data": ["Retained", "Dropped Out", "Retention Rate (%)"]},
    "xAxis": {
      "type": "category",
      "data": df_retention["Semester"].tolist(),
      "axisLabel": {"interval": 0, "rotate": 30},
    },
    "yAxis": [
      {"type": "value", "name": "Number of Students"},
      {"type": "value", "name": "Rate (%)", "min": 0, "max": 100}
    ],
    "series": [
      {
        "name": "Retained",
        "type": "bar",
        "data": df_retention["Retained"].tolist(),
        "itemStyle": {"color": "#10b981"}
      },
      {
        "name": "Dropped Out",
        "type": "bar",
        "data": df_retention["Dropped Out"].tolist(),
        "itemStyle": {"color": "#ef4444"}
      },
      {
        "name": "Retention Rate (%)",
        "type": "line",
        "yAxisIndex": 1,
        "data": df_retention["Retention Rate (%)"].tolist(),
        "itemStyle": {"color": "#3b82f6"}
      }
    ]
  }
  st_echarts(options=option, height="500px")
```

6. Top Performers per Program

```
# ------------------------------
# 6. Top Performers per Program
# ------------------------------
elif report == "6. Top Performers per Program":
    st.subheader("🏆 Top Performers per Program")

    # --- Filters ---
    courses = r2.get_courses() + ["All"]
    year_levels = r2.get_year_levels() + ["All"]

    col1, col2 = st.columns(2)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)

    # --- Apply filters ---
    course_filter = selected_course if selected_course != "All" else None
    year_level_filter = selected_year_level if selected_year_level != "All" else None

    with st.spinner(f"Preparing data for {report}.", show_time = True):
        df_top = r.get_top_performers(course=course_filter, year_level=year_level_filter)  # columns:
['Program', 'Semester', 'Student ID', 'Student Name', 'GPA', 'Rank']

    st.dataframe(df_top)

    st.markdown("""
    **Insight:**
    Highlights **high-achieving students** in each program. Useful for **recognition, awards, and
program benchmarking**.
    """)

    if not df_top.empty:
        # --- Chart ---
        st.markdown("### 📊 Top 5 Performers by GPA per Program")

        # Filter for top 5 in each program
        df_top_5 = df_top[df_top['Rank'] <= 5]
```

```
# Create a chart for each program
programs = df_top_5['Program'].unique()
for program in programs:
    st.markdown(f"#### {program}")
    program_data = df_top_5[df_top_5['Program'] == program]

    option = {
        "tooltip": {"trigger": "axis", "axisPointer": {"type": "shadow"}},
        "xAxis": {
            "type": "category",
            "data": program_data["Student Name"].tolist(),
            "axisLabel": {"interval": 0, "rotate": 30},
        },
        "yAxis": {"type": "value", "name": "GPA", "min": 85},
        "series": [{
            "name": "GPA",
            "type": "bar",
            "data": program_data["GPA"].tolist(),
            "label": {"show": True, "position": "top"},
            "itemStyle": {
                "color": {
                    "type": "linear", "x": 0, "y": 0, "x2": 0, "y2": 1,
                    "colorStops": [
                        {"offset": 0, "color": "#10b981"},
                        {"offset": 1, "color": "#3b82f6"}
                    ]
                }
            }
        }]
    }
    st_echarts(options=option, height="400px", key=f"top_performers_{program}")
```

7. Curriculum Progress and Advising

```
# ------------------------------
# 7. Curriculum Progress Viewer
# ------------------------------
elif report == "7. Curriculum Progress Viewer":
    st.subheader("📚 Curriculum Progress Viewer")

    # --- Step 1: Select Course ---
    courses = sorted(db.students.distinct("Course"))
    selected_course = st.selectbox("Select Course:", courses)

    # --- Step 2: Search Student by Name ---
    search_name = st.text_input("Search Student by Name (wildcard):")
    search_trigger = st.button("Search Student")

    if search_trigger:
        if search_name.strip():
            with st.spinner("Searching please wait...",show_time=True):
                results = r2.student_find(search_name, db.students, course=selected_course)
            if results:
                st.session_state.search_results = results
            else:
                st.warning("No student found.")
                st.session_state.search_results = []
        else:
            st.warning("Please enter a name to search.")
            st.session_state.search_results = []

    # --- Step 3: Show search results as inline rows with select buttons ---
    if "search_results" in st.session_state and st.session_state.search_results:
        st.subheader("Search Results")
        for student in st.session_state.search_results:
            col1, col2 = st.columns([5, 1])
            col1.write(f"{student['Name']} | {student.get('Course','')} | Year Level:
{student.get('YearLevel','')}")
            if col2.button("Select", key=f"select_{student['_id']}"):
                st.session_state.selected_student = student
                st.session_state.search_results = []  # clear search results
                st.rerun()  # reload page to show curriculum
```

```
    # --- Step 4: Display Curriculum if a student is selected ---
    if (
        "selected_student" in st.session_state
        and st.session_state.selected_student
        and search_trigger is False  # only show after explicit selection, not auto-load
    ):
        student = st.session_state.selected_student
        st.markdown(f"""
        <div style="border:2px solid #1E90FF; padding:10px; border-radius:5px;
background-color:#E6F0FF">
        <strong>Name:</strong> {student['Name']}<br>
        <strong>Student ID:</strong> {student['_id']}<br>
        <strong>Course:</strong> {student['Course']}<br>
        <strong>Year Level:</strong> {student.get('YearLevel', '')}<br>
        </div>
        """, unsafe_allow_html=True)

        with st.spinner(f"Loading curriculum for {student['Course']}..."):
            df_curriculum = r.get_curriculum_progress(program=student['Course'])

        if not df_curriculum.empty:
            df_curriculum = df_curriculum.sort_values(["Year", "Semester", "Subject Code"])
            years = df_curriculum['Year'].unique()

            for year in years:
                with st.expander(f"🎓 Year: {year}", expanded=True):
                    year_data = df_curriculum[df_curriculum['Year'] == year]
                    semesters = year_data['Semester'].unique()
                    for semester in semesters:
                        with st.expander(f"📚 Semester: {semester}", expanded=True):
                            sem_data = year_data[year_data['Semester'] == semester]
                            sem_display = sem_data[[
                                "Subject Code", "Subject Description", "Lec Hours",
                                "Lab Hours", "Units", "Prerequisites"
                            ]]
                            st.dataframe(sem_display, width="stretch")
                            total_units = sem_data["Units"].sum()
                            st.markdown(f"**Total Units:** {total_units}")
```

```
        # --- Chart for Curriculum ---
        st.markdown("### 📊 Curriculum Workload Distribution")

        # Group by Year and Semester to sum units
        workload = df_curriculum.groupby(['Year', 'Semester'])['Units'].sum().reset_index()
        workload['Term'] = workload['Year'].astype(str) + ' - ' + workload['Semester'].astype(str)

        option = {
          "tooltip": {"trigger": "axis", "axisPointer": {"type": "shadow"}},
          "xAxis": {
            "type": "category",
            "data": workload["Term"].tolist(),
            "name": "Term"
          },
          "yAxis": {"type": "value", "name": "Total Units"},
          "series": [{
            "name": "Total Units",
            "type": "bar",
            "data": workload["Units"].tolist(),
            "label": {"show": True, "position": "top"},
            "itemStyle": {"color": "#8369CF"}
          }]
        }
        st_echarts(options=option, height="400px")

semester_dict = get_semesters(r2.db)   # returns {_id: "Semester SchoolYear"}
 semester_options = [""] + list(semester_dict.values())

 selected_semester_predict = st.selectbox(
    "Select Semester for Prediction:",
    semester_options
 )

 if selected_semester_predict != "":
    # Get Curriculum & Grades

    curriculum_df = r2.get_curriculum(selected_course)
    student_grades_df = r2.get_student_subjects_grades(student.get("_id"))

    # st.dataframe(curriculum_df)
    # st.dataframe(student_grades_df)
```

```
    if not curriculum_df.empty:
        passed_subjects = []
        if not student_grades_df.empty:
            passed_subjects = student_grades_df[student_grades_df["Grade"] >= 75]["Subject
Code"].tolist()

        curriculum_df.columns = curriculum_df.columns.str.strip()

        taken_subjects = []
        if not student_grades_df.empty and "Subject Code" in student_grades_df.columns:
            taken_subjects = student_grades_df["Subject Code"].tolist()

        print('A ====================================')

        selected_semester, selected_year =  selected_semester_predict.replace("Sem",
"").lower().split()

        potential_subjects = curriculum_df[
            (curriculum_df["semester"].str.lower() == selected_semester)
            & (curriculum_df["year"] == student.get("YearLevel"))
            & (~curriculum_df["Subject Code"].isin(taken_subjects))
        ].copy()
        print('selected_semester_predict:',selected_semester_predict.replace("Sem", "").lower())
        print('YearLevel:',student.get("YearLevel"))
        print('taken_subjects:',taken_subjects)
        available_subjects, blocked_subjects = [], []
        print('B ====================================')
        print('potential_subjects:',potential_subjects)
        for _, row in potential_subjects.iterrows():
            print('available_subjects:',available_subjects)
            prerequisites = row.get("preRequisites", [])
            if not prerequisites or all(prereq in passed_subjects for prereq in prerequisites):
                available_subjects.append(row)
            else:
                blocked_subjects.append(row)

        available_subjects_df = pd.DataFrame(available_subjects)
        blocked_subjects_df = pd.DataFrame(blocked_subjects)

        st.markdown(f"#### ✅ Recommended Subjects for {selected_semester_predict}")
```

```
        if not available_subjects_df.empty:
            st.dataframe(available_subjects_df[["Subject Code", "Description", "unit"]])
        else:
            st.info("No subjects available for enrollment.")

        st.markdown(f"#### 🚫 Blocked Subjects for {selected_semester_predict}")
        if not blocked_subjects_df.empty:
            st.dataframe(blocked_subjects_df[["Subject Code", "Description", "unit", "preRequisites"]])
        else:
            st.info("No blocked subjects for this semester.")
    else:
        st.warning("No curriculum found for this course.")
```

Faculty Dashboard Reports

1. Class Grade Distribution

```
def class_grade_distribution(db):
    user_role = st.session_state.get("user_role", "")
    if user_role == "teacher":
        teacher_name = st.session_state.get("fullname", "")
        teacher_names = [teacher_name]
    else:
        teacher_names = get_teachers(db)
        teacher_name = None

    # with col1:
    if user_role != "teacher":
        teacher_name = st.selectbox("Select Faculty", [""] + list(teacher_names))
    else:
        st.markdown(f"👩‍🏫 Faculty: **{teacher_name}**")

    # with col2:
    semester_dict = get_semesters(db, teacher=teacher_name)
    semester_options = [""] + list(semester_dict.values())
    selected_sem_year = st.selectbox("Select Semester & Year", semester_options)

    # --- Generate report button ---
    if st.button("Generate Report"):
        if not teacher_name:
            st.warning("Please select a faculty member to view the report.")
            return
        if not selected_sem_year:
            st.warning("Please select a semester & year.")
            return

        # Reverse lookup semester ID
        semester_id = next((k for k, v in semester_dict.items() if v == selected_sem_year), None)
        if not semester_id:
            st.warning("Selected semester and school year combination not found.")
            return

        st.markdown("### 📊 Class Grade Distribution")
        st.markdown(f"**Faculty Name:** `{teacher_name}`")
```

```
st.markdown(f"**Semester and School Year:** `{selected_sem_year}`")

# Subjects taught by teacher
teacher_subjects_cursor = db.subjects.find({"Teacher": teacher_name})
teacher_subject_codes = [s["_id"] for s in teacher_subjects_cursor]
if not teacher_subject_codes:
    st.warning(f"No subjects found for teacher: {teacher_name}")
    return

grades_cursor = db.grades.find({
    "Grades": {"$ne": []},
    "SemesterID": semester_id
})

report_data = []
subject_details = {
    s["_id"]: s["Description"]
    for s in db.subjects.find({"_id": {"$in": teacher_subject_codes}})
}
for code, name in subject_details.items():
    report_data.append({
        "Course Code": code,
        "Course Name": name,
        "95-100(%)": 0,
        "90-94 (%)": 0,
        "85-89 (%)": 0,
        "80-84(%)": 0,
        "75-79%": 0,
        "Below 75(%)": 0,
        "student_count": 0,
    })

# Process grades
for grade_entry in grades_cursor:
    for i, subject_code in enumerate(grade_entry.get("SubjectCodes", [])):
        if subject_code in teacher_subject_codes:
            report_row = next(
                (item for item in report_data if item["Course Code"] == subject_code), None
            )
            if report_row:
                grade = grade_entry.get("Grades", [])[i]
```

```
                report_row["student_count"] += 1
                if grade >= 95:
                    report_row["95-100(%)"] += 1
                elif 90 <= grade <= 94:
                    report_row["90-94 (%)"] += 1
                elif 85 <= grade <= 89:
                    report_row["85-89 (%)"] += 1
                elif 80 <= grade <= 84:
                    report_row["80-84(%)"] += 1
                elif 75 <= grade <= 79:
                    report_row["75-79%"] += 1
                else:
                    report_row["Below 75(%)"] += 1

    if not any(row['student_count'] > 0 for row in report_data):
        st.info("No student grade data found for the selected faculty and term.")
        return

    # Build DataFrame
    df = pd.DataFrame(report_data)
    df = df[df['student_count'] > 0].copy()
    grade_cols = ["95-100(%)", "90-94 (%)", "85-89 (%)", "80-84(%)", "75-79%", "Below 75(%)"]

    for col in grade_cols:
        df[col] = (df[col] / df['student_count'] * 100).round(2).astype(str) + '%'

    st.dataframe(df[["Course Code", "Course Name"] + grade_cols])

    # Histogram
    st.markdown("---")
    st.markdown("### Grade Distribution Histograms")

    for _, row in df.iterrows():
        st.markdown(f"#### {row['Course Name']} ({row['Course Code']})")
        plot_values = [float(str(row[col]).replace('%', '')) for col in grade_cols]
        plot_labels = [col.replace('(%)', '').replace('%', '').strip() for col in grade_cols]

        fig, ax = plt.subplots()
        ax.bar(plot_labels, plot_values)
        ax.set_ylabel("Percentage of Students (%)")
        ax.set_title(f"Grade Distribution for {row['Course Code']}")
```

```
        plt.xticks(rotation=45, ha="right")
        st.pyplot(fig)
```

2. Student Progress Tracker

```
def student_progress_tracker_page(db):
    """
    Displays the Student Progress Tracker report page.
    """
    st.markdown("### 📈 Student Progress Tracker")
    st.markdown("Shows longitudinal performance for individual students.")

    # --- Determine teacher list based on user role ---
    user_role = st.session_state.get("user_role", "")
    if user_role == "registrar":
        teacher_list = get_teachers(db)
    elif user_role == "teacher":
        teacher_list = [st.session_state.get("fullname", "")]
    else:
        teacher_list = []

    if not teacher_list:
        st.warning("Faculty name is required.")
        return

    # --- Teacher selectbox ---
    teacher_name = st.selectbox("Select Teacher", teacher_list)

    # --- Load data ---
    with st.spinner("Loading teacher's handled Subjects...", show_time=True):
        year_levels = get_year_levels(db)
        courses = get_courses(db)
        subjects_df = get_subjects_by_teacher(db, teacher_name)

    if not courses or not year_levels or subjects_df.empty:
        st.warning("⚠️ Missing data in one or more collections.")
        return

    # --- Prepare subject list ---
```

```
if "Subject Code" in subjects_df.columns:
    subject_list = [""] + subjects_df["Subject Code"].tolist()
elif "Code" in subjects_df.columns:  # fallback
    subject_list = [""] + subjects_df["Code"].tolist()
else:
    subject_list = []
    st.warning("⚠️ No subject code column found in subjects_df.")

# --- Filters ---
with st.container():
    col1, col2, col3 = st.columns(3)
    with col1:
        selected_course = st.selectbox("Filter by Course", courses)
    with col2:
        selected_year_level = st.selectbox("Filter by Year Level", year_levels)
    with col3:
        selected_subject = st.selectbox("Filter by Subject", subject_list)

# --- Generate report ---
if st.button("Generate Report"):
    with st.spinner("Fetching student progress data...", show_time=True):
        progress_data = get_student_progress_data(
            db, selected_course, selected_year_level, selected_subject
        )

    if progress_data.empty:
        st.warning("No data found for the selected filters.")
        return

    # --- Display DataFrame ---
    st.markdown("### Student GPA Progress")

    def style_trend(val):
        color_map = {
            "Improving": "color: green",
            "Need Attention": "color: red",
            "Stable High": "color: blue",
            "Consistently Low": "color: orange",
            "Stable": "color: black",
            "N/A": "color: grey"
        }
```

```
    return color_map.get(val, "")

display_df = progress_data.fillna('-')
st.dataframe(display_df.style.applymap(style_trend, subset=['Overall Trend']))

# --- Visualizations ---
progress_data = progress_data.drop(columns=['StudentID']).fillna(0)
gpa_cols = progress_data.select_dtypes(include='number').columns.tolist()
avg_per_semester = progress_data[gpa_cols].mean().tolist()

st.markdown("#### Average GPA Trend per Semester")
line_options = {
    "tooltip": {"trigger": "axis"},
    "xAxis": {"type": "category", "data": gpa_cols, "name": "Semester"},
    "yAxis": {"type": "value", "name": "Average GPA"},
    "series": [{
        "name": "Average GPA",
        "type": "line",
        "data": avg_per_semester,
        "smooth": True,
    }],
}
st_echarts(options=line_options, height="400px")

# Scatter chart
st.markdown("#### GPA Progress - Scatter Chart")
progress_data["GeneralAverage"] = progress_data[gpa_cols].mean(axis=1)
all_values = progress_data[gpa_cols + ["GeneralAverage"]].values.flatten()
min_val, max_val = 50, float(pd.Series(all_values).max())
scatter_options = {
    "tooltip": {"trigger": "item", "formatter": "{b}: {c}"},
    "legend": {"data": gpa_cols},
    "xAxis": {"type": "value", "name": "General Average GPA", "min": min_val, "max": max_val},
    "yAxis": {"type": "value", "name": "Semester GPA", "min": min_val, "max": max_val},
    "series": []
}
for col in gpa_cols:
    scatter_options["series"].append({
        "name": col,
        "type": "scatter",
        "data": [
```

```
                {"value": [row["GeneralAverage"], row[col]], "name": row["Name"]}
                for _, row in progress_data.iterrows() if pd.notnull(row[col])
            ],
            "symbolSize": 12,
        })
    st_echarts(options=scatter_options, height="500px")


def get_student_progress_data(db, course, year_level, subject_code):
    """
    Fetches and processes data to generate the student progress report.
    """
    import pandas as pd

    students_df = pd.DataFrame(list(db.students.find()))
    grades_df = pd.DataFrame(list(db.grades.find()))
    subjects_df = pd.DataFrame(list(db.subjects.find()))
    semesters_df = pd.DataFrame(list(db.semesters.find()))

    if grades_df.empty or subjects_df.empty or students_df.empty or semesters_df.empty:
        print("⚠️ One of the collections is empty.")
        return pd.DataFrame()

    students_df = students_df.rename(columns={'_id': 'StudentID'})
    subjects_df = subjects_df.rename(columns={'_id': 'SubjectCode'})
    semesters_df = semesters_df.rename(columns={'_id': 'SemesterID'})

    # filter students
    if course:
        students_df = students_df[students_df['Course'] == course]
    if year_level:
        students_df = students_df[students_df['YearLevel'] == year_level]

    # explode grades
    if isinstance(grades_df.get("SubjectCodes").iloc[0], list):
        grades_exploded = grades_df.explode(['SubjectCodes', 'Grades', 'Teachers'])
    else:
        grades_exploded = grades_df.copy()
    grades_exploded = grades_exploded.rename(columns={'SubjectCodes': 'SubjectCode'})

    # filter subject
```

```
    if subject_code:
        students_who_took_subject = grades_exploded[grades_exploded['SubjectCode'] ==
subject_code]['StudentID'].unique()
        students_df = students_df[students_df['StudentID'].isin(students_who_took_subject)]

    # merge grades with subjects
    grades_with_units = pd.merge(grades_exploded, subjects_df[['SubjectCode', 'Units']],
on='SubjectCode', how="left")
    grades_with_units['GradePoints'] = grades_with_units['Grades'] * grades_with_units['Units']

    gpa_df = grades_with_units.groupby(['StudentID', 'SemesterID']).apply(
        lambda x: pd.Series({'GPA': x['GradePoints'].sum() / x['Units'].sum() if x['Units'].sum() else
None})
    ).reset_index()

    # pivot
    gpa_pivot = gpa_df.pivot(index='StudentID', columns='SemesterID', values='GPA').reset_index()
    semester_map = semesters_df.set_index('SemesterID')[['Semester', 'SchoolYear']].to_dict('index')
    new_columns = {sem_id: f"{sem_info['Semester']} {sem_info['SchoolYear']}" for sem_id, sem_info
in semester_map.items()}
    gpa_pivot = gpa_pivot.rename(columns=new_columns)

    # order columns by SchoolYear and Semester
    semester_rank = {'Spring': 1, 'Summer': 2, 'Fall': 3}  # adjust if you have other semesters
    semester_order = semesters_df.copy()
    semester_order['semester_num'] = semester_order['Semester'].map(semester_rank)
    semester_order = semester_order.sort_values(['SchoolYear', 'semester_num'])
    gpa_cols_ordered = [new_columns[sem_id] for sem_id in semester_order['SemesterID'] if
new_columns[sem_id] in gpa_pivot.columns]

    # keep only semester columns where average GPA > 0
    non_zero_semesters = [c for c in gpa_cols_ordered if gpa_pivot[c].dropna().mean() > 0]

    # final pivot columns
    gpa_pivot = gpa_pivot[['StudentID'] + non_zero_semesters]

    # merge student info
    final_df = pd.merge(students_df[['StudentID', 'Name']], gpa_pivot, on='StudentID', how='inner')

    # redefine gpa_cols for trend calculation
    gpa_cols = non_zero_semesters
```

```
    # calculate trend
    def calculate_trend(row):
        gpas = row[gpa_cols].dropna().values
        if len(gpas) < 2:
            return "N/A"
        avg_gpa = sum(gpas) / len(gpas)
        if all(gpa > 3.5 for gpa in gpas):
            return "Stable High"
        if all(gpa < 2.5 for gpa in gpas):
            return "Consistently Low"
        if gpas[-1] > gpas[0] and gpas[-1] > avg_gpa:
            return "Improving"
        if gpas[-1] < gpas[0] and gpas[-1] < avg_gpa:
            return "Need Attention"
        return "Stable"

    final_df['Overall Trend'] = final_df.apply(calculate_trend, axis=1)

    return final_df
```

3. Subject Difficulty Heatmap

```
def subject_difficulty_page(db):

    st.markdown("### 📊 Subject Difficulty Report")
    st.markdown("Visualizes subjects with high failure or dropout rates handled by the faculty.")

    # teacher from session (or allow manual input for testing)
    teacher_name = st.session_state.get("fullname", "")
    if not teacher_name:
        teacher_name = st.text_input("Teacher full name (for testing)", value="")
        if not teacher_name:
            st.warning("Please set `st.session_state['fullname']` or enter a name for testing.")
            return

    # load collections
    with st.spinner("Loading data...", show_time=True):
        semesters_list = list(db.semesters.find())
```

```
    subjects_df = get_subjects_by_teacher(db, teacher_name)
    grades_list = list(db.grades.find())
    # student info map (for possible future drilldowns)
    students_list = list(db.students.find())

# semesters -> DataFrame + map
if not semesters_list:
    st.warning("No semester data found.")
    return

semesters_df = pd.DataFrame(semesters_list)
if "_id" in semesters_df.columns:
    semesters_df = semesters_df.rename(columns={"_id": "SemesterID"})
for c in ["SemesterID", "Semester", "SchoolYear"]:
    if c not in semesters_df.columns:
        semesters_df[c] = ""

# normalize strings
semesters_df["SemesterID"] = semesters_df["SemesterID"].astype(str)
semesters_df["Semester"] = semesters_df["Semester"].astype(str)
semesters_df["SchoolYear"] = semesters_df["SchoolYear"].astype(str)

semester_labels = (semesters_df["Semester"] + " " + semesters_df["SchoolYear"]).tolist()
# map label -> list of semester ids (some systems may have duplicates)
semester_map = {}
for sid, label in zip(semesters_df["SemesterID"].tolist(), semester_labels):
    semester_map.setdefault(label, []).append(str(sid))

semester_options = ["All Semesters"] + list(semester_map.keys())
selected_semester_label = st.selectbox("📅 Select Semester", semester_options)

# build set of semester ids to include
if selected_semester_label == "All Semesters":
    selected_semester_ids = set(semesters_df["SemesterID"].astype(str).tolist())
else:
    selected_semester_ids = set(semester_map.get(selected_semester_label, []))

if not selected_semester_ids:
    st.info("No semester IDs resolved for selection.")
    return
```

```
    # ensure we have the teacher's subject codes list
    if subjects_df.empty:
        st.info("You have no subjects assigned.")
        return

    subj_codes_handled = set(subjects_df["Subject Code"].astype(str).tolist())
    subj_desc_map = dict(zip(subjects_df["Subject Code"].astype(str),
subjects_df["Description"].astype(str)))

    # build students map for possible lookup
    students_map = {str(s.get("_id")): s for s in students_list}

    # Flatten grades list by zipping parallel arrays — robust and lossless
    flat_rows = []
    for gdoc in grades_list:
        sid = gdoc.get("StudentID")
        sid_s = str(sid) if sid is not None else ""
        sem_raw = gdoc.get("SemesterID", "")
        sem_s = str(sem_raw)

        # if this grade doc's semester is not in the selected set -> skip early
        if sem_s not in selected_semester_ids:
            continue

        subject_codes = gdoc.get("SubjectCodes", []) or []
        grades = gdoc.get("Grades", []) or []
        teachers = gdoc.get("Teachers", []) or []
        status = gdoc.get("Status", []) or []

        # iterate indices, use zip_longest so we won't lose mis-sized arrays (use None for missing)
        for subj, grd, tchr, stat in zip_longest(subject_codes, grades, teachers, status, fillvalue=None):
            subj_s = str(subj).strip() if subj is not None else ""
            tchr_s = str(tchr).strip() if tchr is not None else ""
            # only include entries for this teacher (ensures teacher filter works)
            if tchr_s != teacher_name:
                continue
            # only consider subjects that the teacher handles (extra guard)
            if subj_s not in subj_codes_handled:
                continue

            # coerce grade to numeric if possible, else None
```

```
            try:
                grd_num = float(grd) if grd is not None and str(grd).strip() != "" else None
            except Exception:
                grd_num = None

            flat_rows.append({
                "StudentID": sid_s,
                "SemesterID": sem_s,
                "SubjectCode": subj_s,
                "Grade": grd_num,
                "Status": str(stat) if stat is not None else "",
                "Teacher": tchr_s
            })

    # if no rows found after applying teacher+semester filters
    if not flat_rows:
        st.info("No grade rows found for this teacher in the selected semester(s).")
        return

    flat_df = pd.DataFrame(flat_rows)

    # helper counters
    def count_fails(df_slice, passing_threshold=75):
        if df_slice.empty:
            return 0
        # prefer explicit Status column if it uses e.g. 'Failed'
        if df_slice["Status"].astype(str).str.contains("fail", case=False, na=False).any():
            return df_slice[df_slice["Status"].astype(str).str.contains("fail", case=False,
na=False)]["StudentID"].nunique()
        # else use numeric Grades (< passing threshold)
        if "Grade" in df_slice.columns:
            mask = pd.to_numeric(df_slice["Grade"], errors="coerce") < passing_threshold
            return df_slice[mask]["StudentID"].nunique()
        return 0

    def count_dropouts(df_slice):
        if df_slice.empty:
            return 0
        if df_slice["Status"].astype(str).str.contains("drop", case=False, na=False).any():
            return df_slice[df_slice["Status"].astype(str).str.contains("drop", case=False,
na=False)]["StudentID"].nunique()
```

```
        # fallback: missing grade might indicate dropout (best-effort)
        if "Grade" in df_slice.columns:
            return df_slice[df_slice["Grade"].isna()]["StudentID"].nunique()
        return 0

    # Build summary per subject the teacher handles (even if no rows for some subjects)
    rows = []
    for subj_code in sorted(subj_codes_handled):
        subj_rows = flat_df[flat_df["SubjectCode"].astype(str) == subj_code]
        total_students = subj_rows["StudentID"].nunique() if not subj_rows.empty else 0
        fails = count_fails(subj_rows)
        dropouts = count_dropouts(subj_rows)

        fail_rate = round((fails / total_students) * 100, 2) if total_students else 0.0
        dropout_rate = round((dropouts / total_students) * 100, 2) if total_students else 0.0

        if fail_rate >= 20 or dropout_rate >= 10:
            difficulty = "High"
        elif fail_rate >= 10 or dropout_rate >= 5:
            difficulty = "Medium"
        else:
            difficulty = "Low"

        rows.append({
            "Course Code": subj_code,
            "Course Name": subj_desc_map.get(subj_code, ""),
            "Fail Rate (%)": fail_rate,
            "Dropout Rate (%)": dropout_rate,
            "Difficulty Level": difficulty,
            "Total": total_students
        })

    summary_df = pd.DataFrame(rows)

    # show table
    display_cols = ["Course Code", "Course Name", "Fail Rate (%)", "Dropout Rate (%)", "Difficulty
Level", "Total"]
    st.markdown("### 📋 Subject Difficulty Table")
    st.dataframe(summary_df[display_cols].sort_values(by=["Fail Rate (%)", "Dropout Rate (%)"],
ascending=False).reset_index(drop=True))
```

```
    # build heatmap data in the shape [xName, yName, value] (ECharts accepts category names)
    subject_codes = summary_df["Course Code"].tolist()
    y_categories = ["Fail Rate (%)", "Dropout Rate (%)"]
    heatmap_data = []
    for _, r in summary_df.iterrows():
        heatmap_data.append([r["Course Code"], "Fail Rate (%)", float(r["Fail Rate (%)"])])
        heatmap_data.append([r["Course Code"], "Dropout Rate (%)", float(r["Dropout Rate (%)"])])

    heatmap_options = {
        "tooltip": {
            # no python % formatting here, plain JS string
            "formatter": """function (params) {
                // params.value = [xName, yName, value] when using category names
                var subj = params.value[0] || params.name || '';
                var metric = params.value[1] || '';
                var value = params.value[2] || 0;
                return params.marker + subj + '<br/>' + metric + ': ' + value + '%';
            }"""
        },
        "grid": {"height": "60%", "top": "10%"},
        "xAxis": {"type": "category", "data": subject_codes, "axisLabel": {"rotate": 45, "interval": 0}},
        "yAxis": {"type": "category", "data": y_categories},
        "visualMap": {"min": 0, "max": 100, "calculable": True, "orient": "horizontal", "left": "center",
"bottom": "0"},
        "series": [{
            "name": "Subject Difficulty",
            "type": "heatmap",
            "data": heatmap_data,
            "label": {"show": True},
            "emphasis": {"itemStyle": {"shadowBlur": 10, "shadowColor": "rgba(0,0,0,0.5)"}}
        }]
    }

    st.markdown("### 🔥 Subject Difficulty Heatmap")
    st_echarts(options=heatmap_options, height="480px")

    # CSV download
    csv = summary_df[display_cols].to_csv(index=False).encode("utf-8")
    safe_label = selected_semester_label.replace(" ", "_")
    st.download_button("Download CSV", data=csv,
file_name=f"subject_difficulty_{teacher_name}_{safe_label}.csv", mime="text/csv")
```

4. Intervention Candidates List

```
def intervention_candidates_page(db):
    """📋 Lists students at academic risk (low or missing grades)."""
    st.markdown("### 🚨 Intervention Candidates List")
    st.markdown("Students flagged due to **low grades (<60)** or **missing grades (INC)**.")

    # --- Determine teacher list based on user role ---
    user_role = st.session_state.get("user_role", "")
    if user_role == "registrar":
        # Registrar sees all teachers
        teacher_list = get_teachers(db)
    elif user_role == "teacher":
        # Teacher sees only themselves
        teacher_list = [st.session_state.get("fullname", "")]
    else:
        teacher_list = []

    if not teacher_list:
        st.warning("Faculty name is required.")
        return

    # --- Teacher selectbox ---
    teacher_name = st.selectbox("Select Teacher", teacher_list)

    # --- Load collections ---
    with st.spinner("Loading data..."):
        students_df = pd.DataFrame(list(db.students.find()))
        grades_list = list(db.grades.find())
        semesters_df = pd.DataFrame(list(db.semesters.find()))
        subjects_df = get_subjects_by_teacher(db, teacher_name)

    if students_df.empty or not grades_list or semesters_df.empty or subjects_df.empty:
        st.warning("⚠️ Missing data in one or more collections.")
        return

    # --- Semester Filter ---
    semesters_df = semesters_df.rename(columns={"_id": "SemesterID"})
    semesters_df["SemesterID"] = semesters_df["SemesterID"].astype(str)
```

```
    semesters_df["Label"] = semesters_df["Semester"].astype(str) + " " +
semesters_df["SchoolYear"].astype(str)

    semester_map = dict(zip(semesters_df["SemesterID"], semesters_df["Label"]))
    semester_options = ["All Semesters"] + list(semester_map.values())
    selected_sem = st.selectbox("📅 Select Semester", semester_options)

    if selected_sem == "All Semesters":
        selected_ids = set(semesters_df["SemesterID"])
    else:
        selected_ids = {k for k, v in semester_map.items() if v == selected_sem}

    # --- Normalize ---
    students_df = students_df.rename(columns={"_id": "StudentID"})
    students_df["StudentID"] = students_df["StudentID"].astype(str)
    subj_map = dict(zip(subjects_df["SubjectCode"], subjects_df["Description"]))

    # --- Flatten grades ---
    flat_rows = []
    for gdoc in grades_list:
        sem = str(gdoc.get("SemesterID", ""))
        if sem not in selected_ids:
            continue
        sid = str(gdoc.get("StudentID", ""))

        for subj, grd, tchr in zip_longest(
            gdoc.get("SubjectCodes", []),
            gdoc.get("Grades", []),
            gdoc.get("Teachers", []),
            fillvalue=None
        ):
            if tchr != teacher_name:
                continue
            subj_s = str(subj) if subj else ""
            grade_val = None
            risk_flag = ""

            # Treat 0 or empty as INC
            if grd is None or str(grd).strip() == "" or float(grd) == 0:
                grade_val, risk_flag = "INC", "Missing Grades"
            else:
```

```
            try:
                grade_val = float(grd)
                if grade_val < 60:
                    risk_flag = "At Risk (<60)"
            except:
                grade_val, risk_flag = "INC", "Missing Grades"

        if risk_flag:
            flat_rows.append({
                "StudentID": sid,
                "SubjectCode": subj_s,
                "Grade": grade_val,
                "RiskFlag": risk_flag
            })

if not flat_rows:
    st.info("✅ No at-risk students found for the selected semester(s).")
    return

flat_df = pd.DataFrame(flat_rows)

# --- Merge with student + subject info ---
final_df = pd.merge(flat_df, students_df[["StudentID", "Name"]], on="StudentID", how="left")
final_df["SubjectName"] = final_df["SubjectCode"].map(subj_map)

display_df = final_df[["StudentID", "Name", "SubjectCode", "SubjectName", "Grade", "RiskFlag"]]

# --- Show table ---
st.dataframe(display_df)

# --- Export CSV ---
csv = display_df.to_csv(index=False).encode("utf-8")
sem_label = selected_sem.replace(" ", "_")
st.download_button(
    label="⬇️ Download CSV",
    data=csv,
    file_name=f"intervention_candidates_{teacher_name}_{sem_label}.csv",
    mime="text/csv"
)
```

5. Grade Submission Status

```
def grade_submission_status_page(db):
    st.markdown("### 📊 Grade Submission Status")
    st.markdown("Tracks whether faculty have completely submitted grades for their classes.")

    # --- Determine teacher list based on user role ---
    user_role = st.session_state.get("user_role", "")
    if user_role == "registrar":
        # Registrar sees all teachers
        teacher_list = get_teachers(db)
    elif user_role == "teacher":
        # Teacher sees only themselves
        teacher_list = [st.session_state.get("fullname", "")]
    else:
        teacher_list = []

    if not teacher_list:
        st.warning("Faculty name is required.")
        return

    # --- Teacher selectbox ---
    teacher_name = st.selectbox("Select Teacher", teacher_list)

    if st.button("Search", key="search_btn"):
        # --- Load collections ---
        with st.spinner("Loading data..."):
            students_df = pd.DataFrame(list(db.students.find()))
            grades_list = list(db.grades.find())
            semesters_df = pd.DataFrame(list(db.semesters.find()))
            subjects_df = pd.DataFrame(list(db.subjects.find({"Teacher": teacher_name})))

        if students_df.empty or not grades_list or semesters_df.empty or subjects_df.empty:
            st.warning("⚠️ Missing data in one or more collections.")
            return

        # --- Semester Filter ---
        semesters_df = semesters_df.rename(columns={"_id": "SemesterID"})
        semesters_df["SemesterID"] = semesters_df["SemesterID"].astype(str)
        semesters_df["Label"] = semesters_df["Semester"].astype(str) + " " +
```

```
semesters_df["SchoolYear"].astype(str)

    semester_map = dict(zip(semesters_df["SemesterID"], semesters_df["Label"]))
    semester_options = ["All Semesters"] + list(semester_map.values())
    selected_sem = st.selectbox("📅 Select Semester", semester_options)

    if selected_sem == "All Semesters":
        selected_ids = set(semesters_df["SemesterID"])
    else:
        selected_ids = {k for k, v in semester_map.items() if v == selected_sem}

    # --- Normalize subjects ---
    subjects_df = subjects_df.rename(columns={"_id": "SubjectCode"})
    subjects_df["SubjectCode"] = subjects_df["SubjectCode"].astype(str)

    # --- Build submission rows ---
    submission_rows = []
    for gdoc in grades_list:
        sem = str(gdoc.get("SemesterID", ""))
        if sem not in selected_ids:
            continue
        sid = str(gdoc.get("StudentID", ""))

        for subj, grd, tchr in zip_longest(
            gdoc.get("SubjectCodes", []),
            gdoc.get("Grades", []),
            gdoc.get("Teachers", []),
            fillvalue=None
        ):
            if tchr != teacher_name:
                continue
            submission_rows.append({
                "SemesterID": sem,
                "StudentID": sid,
                "SubjectCode": str(subj) if subj else "",
                "Grade": grd
            })

    if not submission_rows:
        st.info("✅ No grade records found for the selected semester(s).")
        return
```

```
    grades_df = pd.DataFrame(submission_rows)

    # --- Count submissions ---
    def grade_is_submitted(val):
        return val not in [None, "", "0"]

    grouped = grades_df.groupby("SubjectCode").agg(
        SubmittedGrades=("Grade", lambda x: sum(grade_is_submitted(g) for g in x)),
        TotalStudents=("StudentID", "nunique")
    ).reset_index()

    grouped["SubmissionRate"] = (grouped["SubmittedGrades"] / grouped["TotalStudents"] *
100).round(1).astype(str) + "%"

    # --- Merge with subject info ---
    subj_map = dict(zip(subjects_df["SubjectCode"], subjects_df["Description"]))
    grouped["Subject"] = grouped["SubjectCode"].map(subj_map)

    display_df = grouped[["SubjectCode", "Subject", "SubmittedGrades", "TotalStudents",
"SubmissionRate"]]

    # --- Show table ---
    st.dataframe(display_df)

    # --- Export CSV ---
    csv = display_df.to_csv(index=False).encode("utf-8")
    sem_label = selected_sem.replace(" ", "_")
    st.download_button(
        label="⬇️ Download CSV",
        data=csv,
        file_name=f"grade_submission_status_{teacher_name}_{sem_label}.csv",
        mime="text/csv"
    )
```

6. Custom Query Builder

```python
def custom_query_builder_page(db):
    """
    📋 Custom Query Builder
    Allows users to build filtered queries on student grades.
    Example: Show all students with <75 in CS101.
    """

    st.markdown("### 🔎 Custom Query Builder")
    st.markdown("Build a custom query to filter student performance.")

    # --- Determine teacher list based on user role ---
    user_role = st.session_state.get("user_role", "")
    if user_role == "registrar":
        teacher_list = get_teachers(db)
    elif user_role == "teacher":
        teacher_list = [st.session_state.get("fullname", "")]
    else:
        teacher_list = []

    if not teacher_list:
        st.warning("Faculty name is required.")
        return

    # --- Teacher selectbox ---
    teacher_name = st.selectbox("Select Teacher", teacher_list)

    # --- Load collections ---
    with st.spinner("Loading data..."):
        students_df = pd.DataFrame(list(db.students.find()))
        grades_list = list(db.grades.find())
        subjects_df = pd.DataFrame(list(db.subjects.find({"Teacher": teacher_name})))
        semesters_df = pd.DataFrame(list(db.semesters.find()))

    if students_df.empty or not grades_list or subjects_df.empty or semesters_df.empty:
        st.warning("⚠️ Missing data in one or more collections.")
        return

    # --- Normalize ---
    students_df = students_df.rename(columns={"_id": "StudentID"})
    subjects_df = subjects_df.rename(columns={"_id": "SubjectCode"})
    semesters_df = semesters_df.rename(columns={"_id": "SemesterID"})
```

```
    # Build semester labels
    semesters_df["Label"] = semesters_df["Semester"].astype(str) + " " +
semesters_df["SchoolYear"].astype(str)
    semester_map = dict(zip(semesters_df["SemesterID"].astype(str), semesters_df["Label"]))

    # --- Filters ---
    col1, col2, col3, col4 = st.columns(4)
    selected_teacher = teacher_name  # already selected

    with col1:
        selected_sem = st.selectbox("📅 Semester", ["All"] + list(semester_map.values()))
    with col2:
        selected_subject = st.selectbox("📘 Subject", ["All"] +
subjects_df["SubjectCode"].astype(str).tolist())
    with col3:
        operator = st.selectbox("Operator", ["<", ">", "<=", ">=", "<>"])
    with col4:
        grade_value = st.number_input("Grade", min_value=0, max_value=100, value=75)

    # --- Query Execution ---
    if st.button("Run Query"):
        st.info(f"Current Query: **Show all students with {operator} {grade_value}"
                f"{' in ' + selected_subject if selected_subject != 'All' else ''}"
                f"{' for ' + selected_sem if selected_sem != 'All' else ''}"
                f"{' handled by ' + selected_teacher if selected_teacher else ''}**")

        records = []

        with st.spinner("Fetching information based on query.", show_time=True):
            for gdoc in grades_list:
                sem_id = str(gdoc.get("SemesterID", ""))
                sem_label = semester_map.get(sem_id, "Unknown")

                # filter semester
                if selected_sem != "All" and sem_label != selected_sem:
                    continue

                sid = gdoc.get("StudentID")
                for subj, grd, tchr in zip_longest(
                    gdoc.get("SubjectCodes", []),
```

```
            gdoc.get("Grades", []),
            gdoc.get("Teachers", []),
            fillvalue=None
        ):
            subj = str(subj) if subj else ""
            grd_val = float(grd) if grd is not None and str(grd).strip() != "" else None
            tchr_s = str(tchr) if tchr else ""

            # filter teacher
            if selected_teacher and tchr_s != selected_teacher:
                continue

            # filter subject
            if selected_subject != "All" and subj != selected_subject:
                continue

            # apply operator
            if grd_val is None:
                continue
            match = False
            if operator == "<": match = grd_val < grade_value
            elif operator == ">": match = grd_val > grade_value
            elif operator == "<=": match = grd_val <= grade_value
            elif operator == ">=": match = grd_val >= grade_value
            elif operator == "<>": match = grd_val != grade_value

            if match:
                student_row = students_df[students_df["StudentID"] == sid]
                student_name = student_row["Name"].values[0] if not student_row.empty else
"Unknown"

                subj_desc = subjects_df.loc[subjects_df["SubjectCode"] == subj, "Description"]
                subj_name = subj_desc.values[0] if not subj_desc.empty else "Unknown"

                records.append({
                    "StudentID": sid,
                    "Name": student_name,
                    "Subject Code": subj,
                    "Subject": subj_name,
                    "Grade": f"{grd_val:.0f}%"
                })
```

```
    if not records:
        st.warning("⚠️ No matching records found.")
        return

    result_df = pd.DataFrame(records)
    st.success(f"✅ Found {len(result_df)} matching records")
    st.dataframe(result_df)

    # --- Export ---
    csv = result_df.to_csv(index=False).encode("utf-8")
    st.download_button(
        label="⬇️ Download CSV",
        data=csv,
        file_name=f"custom_query_results.csv",
        mime="text/csv"
    )
```

7. Students Grade Analytics (Per Teacher)

```
def get_subjects_by_teacher(db, teacher_name: str) -> pd.DataFrame:
    try:
        grades: List[Dict] = list(db.grades.find({"Teachers": teacher_name}))
    except Exception as e:
        print(f"Error querying grades collection: {e}")
        return pd.DataFrame()

    if not grades:
        return pd.DataFrame()

    # Extract unique subject codes and semester IDs
    all_subject_codes = set()
    all_semester_ids = set()
    for entry in grades:
        if "SubjectCodes" in entry:
```

```
            all_subject_codes.update(entry["SubjectCodes"])
        if "SemesterID" in entry:
            all_semester_ids.add(entry["SemesterID"])

    # Fetch related subjects and semesters
    subjects = list(db.subjects.find({"_id": {"$in": list(all_subject_codes)}}))
    semesters = list(db.semesters.find({"_id": {"$in": list(all_semester_ids)}}))

    # Convert to DataFrames
    grades_df = pd.DataFrame(grades)
    subjects_df = pd.DataFrame(subjects).rename(columns={"_id": "SubjectCodes"})
    semesters_df = pd.DataFrame(semesters).rename(columns={"_id": "SemesterID"})

    # Explode grades for multi-subject entries
    if not grades_df.empty:
        grades_df = grades_df.explode(["SubjectCodes", "Grades", "Teachers", "Status"])

    # Merge all
    merged = pd.merge(grades_df, subjects_df, how="left", on="SubjectCodes")
    merged = pd.merge(merged, semesters_df, how="left", on="SemesterID")

    # Build DisplayName
    merged["DisplayName"] = merged.apply(
        lambda row: f"{row.get('SubjectCodes', '')} - {row.get('Description', '')} "
                    f"({row.get('Semester', '')}, {row.get('SchoolYear', '')})",
        axis=1
    )

    return merged


def get_student_grades(db, teacher_name, subject_code):
    students_df = pd.DataFrame(list(db.students.find()))
    grades_df = pd.DataFrame(list(db.grades.find()))

    if students_df.empty or grades_df.empty:
        return pd.DataFrame()

    students_df = students_df.rename(columns={'_id': 'StudentID'})

    # explode
```

```
    if isinstance(grades_df.get("SubjectCodes").iloc[0], list):
        grades_exploded = grades_df.explode(['SubjectCodes', 'Grades', 'Teachers'])
    else:
        grades_exploded = grades_df.copy()

    grades_exploded = grades_exploded.rename(columns={'SubjectCodes': 'SubjectCode'})

    # filter by teacher + subject
    filtered = grades_exploded[
        (grades_exploded["SubjectCode"] == subject_code) &
        (grades_exploded["Teachers"] == teacher_name)
    ]

    if filtered.empty:
        return pd.DataFrame()

    merged = pd.merge(filtered, students_df, on="StudentID", how="left")

    return merged[["StudentID", "Name", "Course", "YearLevel", "Grades"]]

def get_teachers(db, teacher=None):
    """
    Returns a list of teacher names from the subjects collection.

    :param teacher: Optional string to filter by a specific teacher
    :return: List of teacher names
    """
    query = {}
    if teacher:
        query['Teacher'] = teacher

    # Use distinct to get unique teacher names
    teacher_names = db.subjects.distinct("Teacher", filter=query)
    return teacher_names


# ---------- Main Page ----------
def student_grade_analytics_page(db):
    st.markdown("## 📊 Students Grade Analytics")

    user_role = st.session_state.get("user_role", "")
```

```
teacher_list = []

if user_role == "registrar":
    teacher_list = get_teachers(db)
elif user_role == "teacher":
    teacher_list = get_teachers(db, st.session_state.get("fullname", ""))
else:
    st.warning("⚠️ Unknown user role.")
    return

if not teacher_list:
    st.warning("⚠️ No teachers available for selection.")
    return

teacher_name = st.selectbox("👩‍🏫 Select Teacher", [""] + teacher_list)
if not teacher_name:
    st.info("Please select a teacher to continue.")
    return

# ---------- Load teacher's subjects ----------
with st.spinner("Loading teacher's subjects..."):
    subjects_df = get_subjects_by_teacher(db, teacher_name)

if subjects_df.empty:
    st.warning("⚠️ No subjects found for this teacher.")
    return

subject_options = [""] + subjects_df["DisplayName"].unique().tolist()
subject_map = dict(zip(subjects_df["DisplayName"], subjects_df["SubjectCodes"]))

selected_subject_label = st.selectbox("📘 Select Subject", subject_options)
selected_subject_code = subject_map.get(selected_subject_label, None)
if not selected_subject_code:
    return

# ---------- Fetch student grades ----------
with st.spinner("Fetching grades..."):
    df = get_student_grades(db, teacher_name, selected_subject_code)

if df.empty:
    st.warning("⚠️ No grades found for this subject.")
```

```
    return

# ---------- Summary Stats ----------
st.subheader(f"📑 Grades Summary of Faculty: {teacher_name}")
summary_data = {
    "Mean": round(df["Grades"].mean(), 2),
    "Median": round(df["Grades"].median(), 2),
    "Highest": round(df["Grades"].max(), 2),
    "Lowest": round(df["Grades"].min(), 2),
}
st.table(pd.DataFrame([summary_data]))

# ---------- Histogram (Grade Distribution) ----------
st.subheader(f"📊 Grade Distribution - {selected_subject_label}")
hist_data = df["Grades"].value_counts().sort_index()
hist_options = {
    "tooltip": {"trigger": "axis"},
    "xAxis": {"type": "category", "data": [str(i) for i in hist_data.index.tolist()]},
    "yAxis": {"type": "value"},
    "series": [{"data": [int(v) for v in hist_data.values.tolist()], "type": "bar"}],
}
st_echarts(options=hist_options, height="400px")

# ---------- Pass vs Fail ----------
st.subheader("✅ Pass vs ❌ Fail")
pass_fail_counts = {
    "Pass": int((df["Grades"] >= 75).sum()),
    "Fail": int((df["Grades"] < 75).sum()),
}
pass_fail_options = {
    "tooltip": {"trigger": "axis"},
    "xAxis": {"type": "category", "data": list(pass_fail_counts.keys())},
    "yAxis": {"type": "value"},
    "series": [{
        "data": [
            {"value": int(pass_fail_counts["Pass"]), "itemStyle": {"color": "green"}},
            {"value": int(pass_fail_counts["Fail"]), "itemStyle": {"color": "red"}},
        ],
        "type": "bar",
    }],
}
```

```
    st_echarts(options=pass_fail_options, height="400px")

    # ---------- Student Grades Table ----------
    st.subheader("📝 Student Grades Table")
    df["GradeDisplay"] = df["Grades"].apply(lambda g: "INC" if g == 0 else str(g))
    df["Status"] = df["Grades"].apply(lambda x: "Pass" if x >= 75 else "Fail")

    def style_pass_fail(val):
        return "color: green; font-weight: bold" if val == "Pass" else "color: red; font-weight: bold"

    st.dataframe(
        df[["StudentID", "Name", "Course", "YearLevel", "GradeDisplay", "Status"]]
        .style.applymap(style_pass_fail, subset=["Status"])
    )
```

Students Dashboard Reports

1. Academic Transcript Viewer

```
def transcript_page(db):
    r = dh.data_helper({"db": db})

    user_role = st.session_state.get("user_role", "")
    StudentID = st.session_state.get("uid")

    st.title("👩‍🎓 Student Transcript & GPA")

    # ---------- ACTION BAR (TOP) ----------
    action_col1, action_col2 = st.columns([1, 1])

    with action_col1:
        if st.button("🖨️ Print Page"):
            st.markdown("<script>window.print()</script>", unsafe_allow_html=True)

    with action_col2:
        pdf_download_placeholder = st.empty()

    # ---------- Student Selection ----------
    if user_role == "registrar":
```

```
    # registrar can select any student
    students = get_students_info(db)   # all students
    if students.empty:
        st.warning("No students found.")
        return

    selected_student = st.selectbox("Select Student", students["Name"].tolist())
    student_row = students[students["Name"] == selected_student].iloc[0]

else:
    # student role: only their own record
    students = get_students_info(db, StudentID=StudentID)
    if students.empty:
        st.warning("No student record found.")
        return

    student_row = students.iloc[0]

# ---------- Display student info ----------
st.write(f"**Student ID:** {student_row['_id']}")
st.write(f"**Name:** {student_row['Name']}")
st.write(f"**Course:** {student_row['Course']}")
# st.write(f"**Year Level:** {student_row['YearLevel']}")

# --- Get Curriculum and Grades ---
student_id = student_row["_id"]
program_code = student_row['Course']

curriculum_df = r.get_curriculum(program_code)
stud_grades = get_student_subjects_grades(db, StudentID=student_id)

if curriculum_df.empty:
    st.warning(f"No curriculum found for the program: {program_code}")
    if not stud_grades.empty:
        st.subheader("Grades")
        st.dataframe(stud_grades)
    return

# --- Merge curriculum with grades ---
if not stud_grades.empty:
    grades_to_merge = stud_grades[['Subject Code', 'Grade', 'SchoolYear', 'Semester']].copy()
```

```
    prospectus_df = pd.merge(curriculum_df, grades_to_merge, on="Subject Code", how="left")
else:
    prospectus_df = curriculum_df.copy()
    prospectus_df['Grade'] = np.nan
    prospectus_df['SchoolYear'] = ''
    prospectus_df['Semester'] = ''

# --- Status Column ---
def grade_status(grade):
    if pd.isna(grade) or grade == '' or grade == 0:
        return "-"
    elif float(grade) >= 75:
        return "Pass"
    else:
        return "Fail"

prospectus_df["Status"] = prospectus_df["Grade"].apply(grade_status)

# --- Styling Functions ---
def color_status(val):
    if val == "Pass":
        return "color: green; font-weight: bold;"
    elif val == "Fail":
        return "color: red; font-weight: bold;"
    else:
        return ""

def color_grade(val):
    if val == "-" or pd.isna(val):
        return ""
    elif float(val) >= 75:
        return "color: green;"
    else:
        return "color: red;"

# --- Display Prospectus per Year/Semester ---
prospectus_df.sort_values(by=['year', 'semester'], inplace=True)

for (year, sem), group in prospectus_df.groupby(["year", "semester"]):
    st.subheader(f"📘 Year {year} / {sem} Semester")
    display_df = group[["Subject Code", "Description", "Grade", "Status", "unit",
```

```
"preRequisites"]].copy()

    # Format Grade column
    display_df['Grade'] = display_df['Grade'].apply(lambda x: f"{x:.2f}" if pd.notna(x) else "-")
    display_df.rename(columns={"unit": "Units", "preRequisites": "Prerequisites"}, inplace=True)

    # --- Apply styling ---
    styled_df = display_df.style.applymap(color_status, subset=["Status"]) \
                        .applymap(color_grade, subset=["Grade"])

    # Display in Streamlit — remove .reset_index()
    st.dataframe(styled_df)

  # --- Calculations for subjects with grades ---
  graded_subjects = prospectus_df.dropna(subset=['Grade'])

  if not graded_subjects.empty:
    # Overall GPA
    gpa = graded_subjects["Grade"].astype(float).mean()
    st.markdown(
      f"<h2 style='text-align: center; color: #2E86C1;'>Overall GPA: {gpa:.2f}</h2>",
      unsafe_allow_html=True
    )

    # GPA Trend
    st.subheader("📈 GPA Trend per Semester")
    gpa_trend = (
      graded_subjects.groupby(["SchoolYear", "Semester"])["Grade"]
      .mean()
      .reset_index()
    )
    gpa_trend["Period"] = gpa_trend["SchoolYear"].astype(str) + " / " +
gpa_trend["Semester"].astype(str)
    gpa_trend.sort_values("Period", inplace=True)

    fig, ax = plt.subplots(figsize=(10, 4))
    ax.plot(gpa_trend["Period"], gpa_trend["Grade"], marker='o', linestyle='-')
    ax.set_ylim(50, 100)
    ax.set_xlabel("Semester")
    ax.set_ylabel("Average Grade")
    ax.set_title("GPA Trend per Semester")
```

```
        ax.grid(True)
        plt.xticks(rotation=45)
        st.pyplot(fig)

        chart_buf = BytesIO()
        fig.savefig(chart_buf, format="png", bbox_inches="tight")
        chart_buf.seek(0)
    else:
        st.info("No grades available to calculate GPA or show trend.")
        gpa = None
        gpa_trend = pd.DataFrame()
        chart_buf = None


    # ---------- PDF GENERATION ----------
    def create_prospectus_pdf(student_row, prospectus_df, gpa, gpa_trend, chart_buf):
        buf = BytesIO()
        doc = SimpleDocTemplate(buf, pagesize=A4, rightMargin=30, leftMargin=30, topMargin=36,
bottomMargin=30)
        styles = getSampleStyleSheet()
        story = []

        # Header
        story.append(Paragraph(f"Student Prospectus — {student_row['Name']}", styles["Title"]))
        story.append(Paragraph(f"Course: {student_row['Course']}    Year Level:
{student_row['YearLevel']}", styles["Normal"]))
        story.append(Paragraph(f"Student ID: {student_row['_id']}", styles["Normal"]))
        story.append(Spacer(1, 12))

        # Summary
        story.append(Paragraph("Prospectus Summary", styles["Heading2"]))
        total_subjects = len(prospectus_df)
        passed_subjects = (prospectus_df['Status'] == 'Pass').sum()
        failed_subjects = (prospectus_df['Status'] == 'Failed').sum()
        not_taken = total_subjects - passed_subjects - failed_subjects

        sum_data = [
            ["Overall GPA", f"{gpa:.2f}" if gpa else "—"],
            ["Total Subjects in Curriculum", total_subjects],
            ["Passed", passed_subjects],
            ["Failed", failed_subjects],
```

```
        ["Not Yet Taken", not_taken],
    ]
    t_sum = Table(sum_data, hAlign="LEFT", colWidths=[180, 200])
    t_sum.setStyle(TableStyle([
        ("GRID", (0, 0), (-1, -1), 0.3, colors.grey),
        ("BACKGROUND", (0, 0), (-1, 0), colors.HexColor("#eef2ff")),
        ("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
    ]))
    story.append(t_sum)
    story.append(Spacer(1, 14))

    # Per-Semester Tables
    for (year, sem), group in prospectus_df.groupby(["year", "semester"]):
        story.append(Paragraph(f"Year {year} / {sem} Semester", styles["Heading2"]))

        data = [["Code", "Description", "Units", "Grade", "Status"]]
        for _, r in group.iterrows():
            grade_str = f"{r['Grade']:.2f}" if pd.notna(r['Grade']) else "—"
            data.append([r["Subject Code"], r["Description"], r["unit"], grade_str, r["Status"]])

        tbl = Table(data, hAlign="LEFT", colWidths=[70, 220, 40, 50, 70])
        tbl.setStyle(TableStyle([
            ("BACKGROUND", (0, 0), (-1, 0), colors.HexColor("#26364a")),
            ("TEXTCOLOR", (0, 0), (-1, 0), colors.whitesmoke),
            ("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
            ("GRID", (0, 0), (-1, -1), 0.3, colors.grey),
            ("ALIGN", (2, 1), (-1, -1), "CENTER"),
        ]))
        story.append(tbl)
        story.append(Spacer(1, 16))

    # GPA Trend
    if chart_buf:
        story.append(Paragraph("GPA Trend Chart", styles["Heading2"]))
        story.append(Image(chart_buf, width=400, height=180))

    doc.build(story)
    buf.seek(0)
    return buf.read()

# --- Download Button ---
```

```
    pdf_buffer = create_prospectus_pdf(student_row, prospectus_df, gpa, gpa_trend, chart_buf)
    if pdf_buffer:
        pdf_download_placeholder.download_button(
            "📄 Download Prospectus (PDF)",
            data=pdf_buffer,
            file_name=f"prospectus_{student_row['_id']}.pdf",
            mime="application/pdf"
        )
```

2. Performance Trend Over Time

```
def performance_trend_page(db):
    r = dh.data_helper({"db": db})
    st.title("📈 Performance Trend Over Time")

    user_role = st.session_state.get("user_role", "")
    StudentID = st.session_state.get("uid", None)

    # --- Student Selection ---
    if user_role == "registrar":
        students = r.get_students()  # all students
        if students.empty:
            st.warning("No students found.")
            return

        student_display = students["Name"] + " (" + students["_id"].astype(str) + ")"
        selected_student = st.selectbox("Select Student", student_display.tolist())
        selected_id = students.iloc[student_display[student_display ==
selected_student].index[0]]["_id"]

        student_info = students[students["_id"] == selected_id]
    else:
        # Student user
        if not StudentID:
            st.warning("Student not logged in.")
            return
        student_info = r.get_students(StudentID=StudentID)

    if student_info.empty:
```

```
        st.warning("Student not found.")
        return

    student_name = student_info.iloc[0]["Name"]
    sid = student_info.iloc[0]["_id"]

    st.subheader(f"Student: {student_name} ({sid})")

    # --- Get Grades ---
    stud_grades = r.get_student_subjects_grades(StudentID=sid)

    if stud_grades.empty or 'Grade' not in stud_grades.columns or stud_grades['Grade'].isnull().all():
        st.info("No grades available to show performance trend.")
        return

    # --- Calculate GPA Trend ---
    stud_grades['Grade'] = pd.to_numeric(stud_grades['Grade'], errors='coerce')

    gpa_trend = (
        stud_grades.dropna(subset=['Grade'])
        .groupby(["SchoolYear", "Semester"])["Grade"]
        .mean()
        .reset_index()
    )

    if gpa_trend.empty:
        st.info("No semester data available to show trend.")
        return

    gpa_trend["Period"] = gpa_trend["SchoolYear"].astype(str) + " / " +
gpa_trend["Semester"].astype(str)
    gpa_trend.sort_values(["SchoolYear", "Semester"], inplace=True)
    gpa_trend.rename(columns={"Grade": "Semester GPA"}, inplace=True)

    # --- Display Chart ---
    fig, ax = plt.subplots(figsize=(10, 5))
    ax.plot(gpa_trend["Period"], gpa_trend["Semester GPA"], marker='o', linestyle='-', color='b')
    ax.set_ylim(50, 100)
    ax.set_xlabel("Semester")
    ax.set_ylabel("Semester GPA")
    ax.set_title("GPA Trend per Semester")
```

```
    ax.grid(True, linestyle='--', alpha=0.6)
    plt.xticks(rotation=45)
    st.pyplot(fig)

    # --- Display Table ---
    st.subheader("Data")
    display_df = gpa_trend[["Period", "Semester GPA"]].copy()
    display_df["Semester GPA"] = display_df["Semester GPA"].map("{:.2f}".format)
    st.dataframe(display_df.rename(columns={"Period": "Semester"}).reset_index(drop=True))
```

3. Subject Difficulty Ratings

```
def calculate_difficulty(grades_series):
    """Calculates difficulty level based on grade distribution."""
    average_grade = grades_series.mean()
    if average_grade >= 85:
        return "Low"
    elif average_grade >= 75:
        return "Medium"
    else:
        return "High"

def subject_difficulty_page(db):
    r = dh.data_helper({"db": db})
    st.title("📊 Subject Difficulty Ratings")

    # --- Role & UID ---
    role = st.session_state.get("user_role", "student")  # e.g. "student" or "registrar"
    StudentID = st.session_state.get("uid", None)

    if role == "registrar":
        students = r.get_students()
        if students.empty:
            st.warning("No students found.")
            return

        selected_student = st.selectbox(
            "Select Student",
            students["Name"].tolist()
```

```
        )
        student_row = students[students["Name"] == selected_student].iloc[0]
        StudentID = student_row["_id"]
        student_name = student_row["Name"]

    elif role == "student":
        if not StudentID:
            st.warning("Student not logged in.")
            return

        student_info = r.get_students(StudentID=StudentID)
        if student_info.empty:
            st.warning("Student not found.")
            return
        student_name = student_info.iloc[0]["Name"]

    else:
        st.error("Unauthorized role.")
        return

    # --- Show selected student ---
    st.subheader(f"Student: {student_name} ({StudentID})")

    # --- Get student's taken subjects and grades ---
    student_grades = r.get_student_subjects_grades(StudentID=StudentID)
    if student_grades.empty:
        st.info("No grades recorded for this student.")
        return

    # Get all subjects for description mapping
    all_subjects_df = r.get_subjects()
    subject_map = {}
    if not all_subjects_df.empty:
        subject_map = all_subjects_df.set_index('Subject Code')['Description'].to_dict()

    # --- Process each subject ---
    report_data = []

    subjects_taken = student_grades[['Subject Code', 'Grade']].drop_duplicates(subset=['Subject
Code'])
```

```
for _, row in subjects_taken.iterrows():
    subject_code = row['Subject Code']
    your_grade = row['Grade']

    all_grades_subject = r.get_all_grades_for_subject(subject_code)
    if all_grades_subject.empty:
        continue

    total_students = len(all_grades_subject)
    grades = all_grades_subject['Grade'].astype(float)

    dist_90_100 = (grades >= 90).sum() / total_students * 100
    dist_80_89 = ((grades >= 80) & (grades < 90)).sum() / total_students * 100
    dist_70_79 = ((grades >= 70) & (grades < 80)).sum() / total_students * 100
    dist_60_69 = ((grades >= 60) & (grades < 70)).sum() / total_students * 100
    dist_lt_60 = (grades < 60).sum() / total_students * 100

    difficulty = calculate_difficulty(grades)

    report_data.append({
        "Subject Code": subject_code,
        "Subject": subject_map.get(subject_code, "N/A"),
        "Total Students": total_students,
        "Your Grade(%)": f"{your_grade:.2f}",
        "90-100% (%)": f"{dist_90_100:.1f}",
        "80-89% (%)": f"{dist_80_89:.1f}",
        "70-79% (%)": f"{dist_70_79:.1f}",
        "60-69% (%)": f"{dist_60_69:.1f}",
        "< 60% (%)": f"{dist_lt_60:.1f}",
        "Difficulty Level": difficulty,
    })

# --- Display Table ---
if not report_data:
    st.info("Could not generate difficulty ratings for this student's subjects.")
    return

report_df = pd.DataFrame(report_data)
st.dataframe(report_df)
```

4. Comparison with Class Average

```
def get_remarks(your_grade, class_average):
    """Generates remarks based on grade vs. class average."""
    diff = your_grade - class_average
    if diff > 10:
        return "Above class average - excellent standing"
    elif diff > 0:
        return "Slightly above average - solid performance"
    elif diff == 0:
        return "At class average"
    elif diff > -10:
        return "Slightly below average - room for improvement"
    else:
        return "Below class average - needs additional support"

def peer_comparison_page(db):
    r = dh.data_helper({"db": db})
    st.title("👫 Subject Peer Comparison")

    # --- Role & UID ---
    role = st.session_state.get("user_role", "student")  # "student" or "registrar"
    StudentID = st.session_state.get("uid", None)

    if role == "registrar":
        students = r.get_students()
        if students.empty:
            st.warning("No students found.")
            return

        selected_student = st.selectbox(
            "Select Student",
            students["Name"].tolist()
        )
        student_row = students[students["Name"] == selected_student].iloc[0]
        StudentID = student_row["_id"]
        student_name = student_row["Name"]

    elif role == "student":
        if not StudentID:
```

```
            st.warning("Student not logged in.")
            return

        student_info = r.get_students(StudentID=StudentID)
        if student_info.empty:
            st.warning("Student not found.")
            return
        student_name = student_info.iloc[0]["Name"]

    else:
        st.error("Unauthorized role.")
        return

    st.subheader(f"Student: {student_name} ({StudentID})")

    # --- Get student's taken subjects and grades ---
    student_grades = r.get_student_subjects_grades(StudentID=StudentID)
    if student_grades.empty:
        st.info("No grades recorded for this student.")
        return

    # Get all subjects for description mapping
    all_subjects_df = r.get_subjects()
    subject_map = all_subjects_df.set_index('Subject Code')['Description'].to_dict() if not
all_subjects_df.empty else {}

    # --- Process each subject ---
    report_data = []
    subjects_taken = student_grades[['Subject Code', 'Grade']].drop_duplicates(subset=['Subject
Code'])

    for _, row in subjects_taken.iterrows():
        subject_code = row['Subject Code']
        your_grade = float(row['Grade'])

        all_grades_subject = r.get_all_grades_for_subject(subject_code)
        if all_grades_subject.empty or len(all_grades_subject) < 2:
            continue  # Not enough data for comparison

        grades = all_grades_subject['Grade'].astype(float)
        total_students = len(grades)
```

```
        class_average = grades.mean()

        # --- Rank calculation ---
        sorted_grades = grades.sort_values(ascending=False)
        # Rank is position (1 = top rank)
        your_rank = (sorted_grades >= your_grade).sum()

        remarks = get_remarks(your_grade, class_average)

        report_data.append({
            "Subject Code": subject_code,
            "Subject": subject_map.get(subject_code, "N/A"),
            "Total Students": total_students,
            "Your Grade(%)": f"{your_grade:.2f}%",
            "Class Average(%)": f"{class_average:.2f}%",
            "Your Rank": f"{your_rank} of {total_students}",
            "Remarks": remarks,
        })

    # --- Display Table ---
    if not report_data:
        st.info("Could not generate peer comparison for this student's subjects.")
        return

    report_df = pd.DataFrame(report_data)
    st.dataframe(report_df)
```

5. Passed vs Failed Summary

```
def get_student_subjects_grades(db, StudentID=None, limit=1000):
    """
    Returns all subjects and grades for a specific student with:
    ["Subject Code", "Description", "Grade", "Semester", "SchoolYear"]
    """

    def query():
        if StudentID is None:
            return pd.DataFrame()

        student_id = int(StudentID)
```

```
        grade_docs = db.grades.find({"StudentID": student_id})  # ✅ Multiple documents

        rows = []
        for grade_doc in grade_docs:
            subject_codes = grade_doc.get("SubjectCodes", [])
            grades = grade_doc.get("Grades", [])
            semester_id = grade_doc.get("SemesterID")

            # Fetch semester info
            sem = db.semesters.find_one({"_id": semester_id})
            semester = sem["Semester"] if sem else None
            school_year = sem["SchoolYear"] if sem else None

            # Process each subject
            for code, grade in zip(subject_codes, grades):
                subj = db.subjects.find_one({"_id": code})
                desc = subj["Description"] if subj else None

                rows.append({
                    "Subject Code": code,
                    "Description": desc,
                    "Grade": grade,
                    "Semester": semester,
                    "SchoolYear": school_year
                })

        # Apply limit
        if limit:
            rows = rows[:limit]

        return pd.DataFrame(rows)

    return query()  # no caching


def passed_failed_summary_page(db):
    r = dh.data_helper({"db": db})
    st.title("📊 Passed vs Failed Summary")

    # --- Role & UID ---
    role = st.session_state.get("user_role", "student")
```

```
StudentID = st.session_state.get("uid", None)

if role == "registrar":
    students = r.get_students()
    if students.empty:
        st.warning("No students found.")
        return

    selected_student = st.selectbox(
        "Select Student",
        students["Name"].tolist()
    )
    student_row = students[students["Name"] == selected_student].iloc[0]
    StudentID = student_row["_id"]
    student_name = student_row["Name"]
    program_code = student_row["Course"]

elif role == "student":
    if not StudentID:
        st.warning("Student not logged in.")
        return

    student_info = r.get_students(StudentID=StudentID)
    if student_info.empty:
        st.warning("Student not found.")
        return
    student_name = student_info.iloc[0]["Name"]
    program_code = student_info.iloc[0]["Course"]

else:
    st.error("Unauthorized role.")
    return

st.subheader(f"Student: {student_name} ({StudentID})")

# --- Get Curriculum and Grades ---
curriculum_df = r.get_curriculum(program_code)
if curriculum_df.empty:
    st.warning(f"No curriculum found for course: {program_code}")
    return
total_required_subjects = len(curriculum_df)
```

```
    stud_grades = get_student_subjects_grades(db,StudentID=StudentID)

    # --- Calculations ---
    if not stud_grades.empty:
        merged_df = pd.merge(
            curriculum_df,
            stud_grades[['Subject Code', 'Grade']],
            on="Subject Code",
            how="left"
        )
        # print('curriculum_df:',curriculum_df)
        print('stud_grades:',stud_grades)
        # print('merged_df:',merged_df)
        merged_df['Grade'] = pd.to_numeric(merged_df['Grade'], errors='coerce')

        passed_subjects = merged_df[merged_df['Grade'] >= 75].shape[0]
        failed_subjects = merged_df[(merged_df['Grade'] < 75) &
(merged_df['Grade'].notna())].shape[0]
        taken_subjects = passed_subjects + failed_subjects
        not_yet_taken = total_required_subjects - taken_subjects
    else:
        passed_subjects = 0
        failed_subjects = 0
        not_yet_taken = total_required_subjects

    # --- Prepare Data for Display ---
    summary_data = {
        "Category": ["Passed Subjects", "Failed Subjects", "Not Yet Taken", "Total Required Subjects"],
        "Count": [passed_subjects, failed_subjects, not_yet_taken, total_required_subjects],
        "Percentage (%)": [
            (passed_subjects / total_required_subjects) * 100 if total_required_subjects > 0 else 0,
            (failed_subjects / total_required_subjects) * 100 if total_required_subjects > 0 else 0,
            (not_yet_taken / total_required_subjects) * 100 if total_required_subjects > 0 else 0,
            100
        ],
        "Description": [
            f"Courses where {student_name} achieved passing grades.",
            f"Courses where {student_name} earned failing grades.",
            "Remaining required courses yet to be taken.",
            "Total courses in the curriculum."
```

```
        ]
    }
    summary_df = pd.DataFrame(summary_data)
    summary_df["Percentage (%)"] = summary_df["Percentage (%)"].map("{:.1f}%".format)

    st.subheader(f"Subject Completion Overview (out of {total_required_subjects} required
subjects)")
    st.dataframe(summary_df)

    # --- Display Bar Chart ---
    st.subheader("Visual Summary")
    chart_data = {
        "Passed": passed_subjects,
        "Failed": failed_subjects,
        "Not Yet Taken": not_yet_taken,
    }

    fig, ax = plt.subplots()
    ax.bar(chart_data.keys(), chart_data.values(), color=['green', 'red', 'grey'])
    ax.set_ylabel("Number of Subjects")
    ax.set_title("Subject Completion Status")

    for i, v in enumerate(chart_data.values()):
        ax.text(i, v + 0.1, str(v), ha='center', fontweight='bold')

    st.pyplot(fig)
```

Recommendations for Dashboard Enhancements

To evolve the Distributed Academic Performance Analytics System beyond descriptive analytics and into predictive and prescriptive intelligence, the following enhancements are recommended:

A. Registrar Dashboard

Predictive Features

1. **Enrollment Demand Forecasting** – Predict enrollment trends by program and semester, identifying courses at risk of over- or under-enrollment.
2. **Graduation Pipeline Projection** – Estimate number of students on track to graduate each term.
3. **Attrition & Retention Risk Modeling** – Forecast which cohorts are likely to have high dropout rates.
4. **Faculty Load Forecast** – Predict teaching load requirements for upcoming semesters.

Prescriptive Features

1. **Course Offering Optimization** – Recommend opening new sections or consolidating low-demand subjects.
2. **Resource Allocation Guidance** – Suggest optimal distribution of faculty, classrooms, and labs based on demand forecasts.
3. **Targeted Retention Programs** – Recommend scholarships, advising, or support interventions for high-risk groups.
4. **Policy Simulation Tools** – Allow “what-if” scenarios (e.g., impact of increasing admission slots or scholarship funds)

B. Faculty Dashboard

Predictive Features

1. **Student Grade Forecasting** – Predict likely final grades for each student early in the semester.
2. **Risk-of-Failure Alerts** – Identify students at medium/high risk of failing or dropping the course.
3. **Class Performance Forecast** – Predict pass rates and grade distributions for the subject.
4. **Bottleneck Topic Prediction** – Forecast which upcoming lessons are likely to lower performance.

Prescriptive Features

1. **Intervention Recommendations** – Suggest targeted actions (e.g., tutoring, review sessions, flexible assessments) for at-risk students.

2. **Adaptive Teaching Suggestions** – Recommend pacing changes or supplementary resources for difficult topics.

3. **Feedback Prioritization** – Highlight which students need one-on-one attention.

4. **Course Design Scenarios** – Allow simulation of grading weight changes or new assessment methods to see their impact on predicted outcomes.

C. Student Dashboard

Predictive Features

1. **Personal Grade Forecasting** – Show predicted final grade ranges for each current subject.

2. **GPA & Graduation Timeline Projection** – Estimate if the student is on track to graduate on time.

3. **Risk Alerts** – Warn students if they are likely to fail a subject or drop below required GPA/scholarship thresholds.

4. **Performance Simulation** – "What-if" tool to see GPA impact of improving or failing certain subjects.

Prescriptive Features

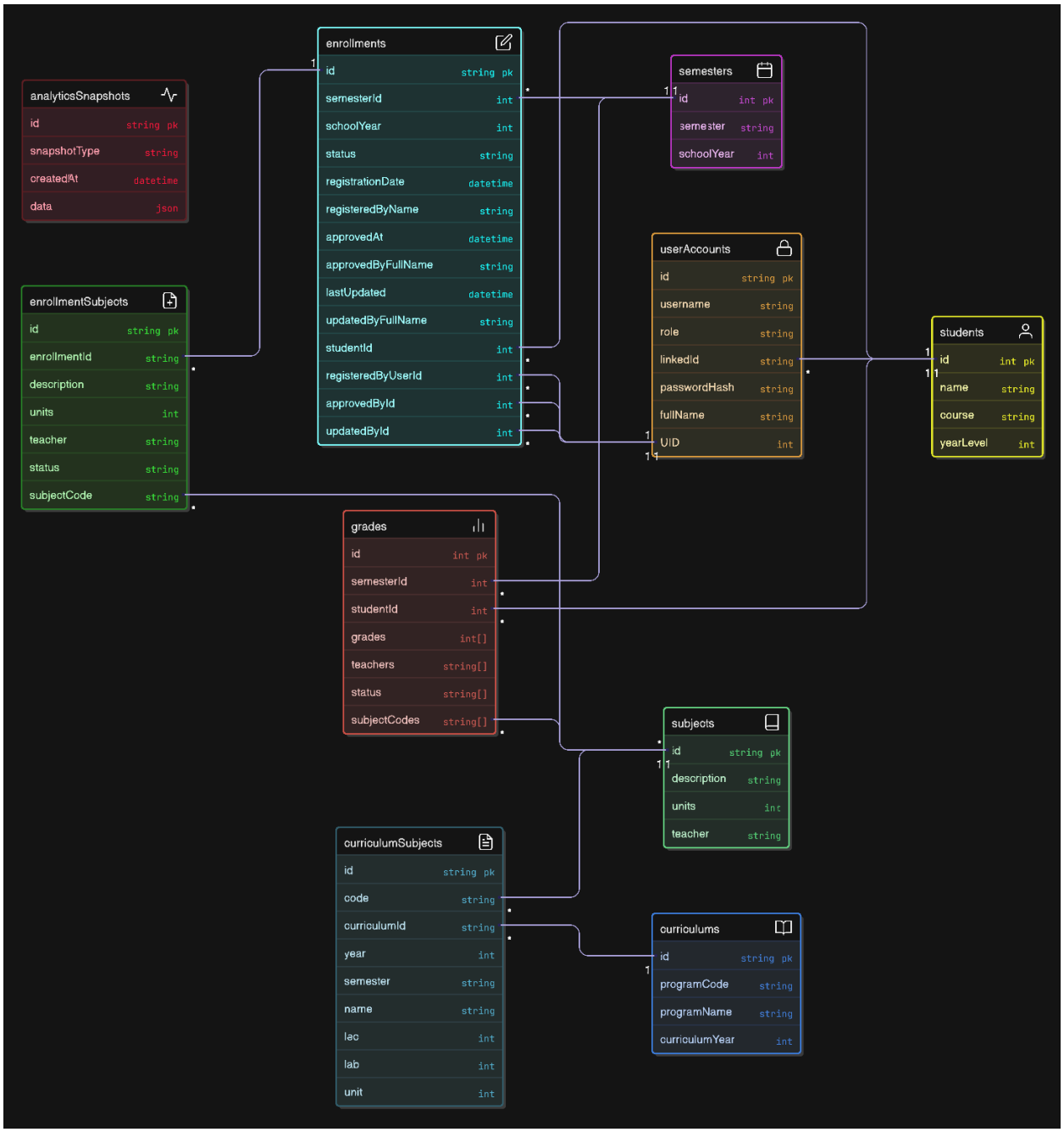
1. **Personalized Study Plan** – Recommend remedial activities, tutoring, or additional practice for weak areas.

2. **Course Planning Assistance** – Suggest optimal subject loads and electives for the next semester.

3. **Success Nudges & Alerts** – Automated reminders about deadlines, attendance, and missing work.

4. **Career/Program Alignment Guidance** – Suggest courses or focus areas aligned with predicted strengths and career goals.

MongoDB Database Model for Dashboard Enhancements



A. Registrar Dashboard – Data Flow

1. enrollments

- Tracks which students are enrolled in what semester, who approved them, etc.
- Used to count enrollments per subject/program.

2. semesters

- Defines academic terms (e.g., Semester 1, 2025–2026).
- Used to group data (enrollment, grades) by time period.

3. curriculums and curriculumSubjects

- Stores program information (e.g., BSCS 2022).
- Ensures students are following correct academic paths.

4. analyticsSnapshots

- Contains precomputed data like trends, bottlenecks, etc. (data field is JSON).
- Helps in forecasting and planning (e.g., which subjects are in high demand).

Registrar used this to:

- Monitor enrollment trends.
- Check compliance with curriculum.
- Predict future subject demand or bottlenecks.

B. Faculty Dashboard – Data Flow

1. grades

- Records students' grades per subject and semester.
- Helps visualize class performance, like pass rates or averages.

2. subjects

- Contains subject info and assigned teacher.
- Used to display what subjects a teacher is handling.

3. enrollmentsSubjects (via enrollments)

- Shows which students are in each subject under a teacher.

4. analyticsSnapshots

- Provides **early warnings** (e.g., failing trends, drop-out risk).

Faculty used this to:

- See their current teaching load.
- Monitor student performance.
- Intervene early for struggling students.

C. Student Dashboard – Data Flow

1. grades

- Displays a student's personal grades and GPA.

2. enrollments

- Lists current and past enrolled subjects and status.

3. curriculums and curriculumSubjects

- Shows required subjects, and highlights what’s completed or pending.

4. analyticsSnapshots

- Provides personalized predictions (GPA trends, risk alerts).

Students used this to:

- Track academic progress.
- Get advice on how to stay on track.
- See what’s left for graduation.