

---

# DanceSoft

---

## Senior Design Final Documentation

DanceSoft

Marcus Berger

Dicheng Wu

May 1, 2016



---

# Contents

---

<b>Title</b>	<b>i</b>
<b>Contents</b>	<b>xi</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xv</b>
<b>List of Algorithms</b>	<b>xvii</b>
<b>Overview Statements</b>	<b>xix</b>
0.1 Mission Statement . . . . .	xix
0.2 Elevator Pitch . . . . .	xix
<b>Document Preparation and Updates</b>	<b>xxi</b>
<b>1 Overview and concept of operations</b>	<b>1</b>
1.1 Team Members and Team Name . . . . .	1
1.2 Client . . . . .	1
1.3 Project . . . . .	1
1.3.1 Purpose of the System . . . . .	1
1.4 Business Need . . . . .	2
1.5 Deliverables . . . . .	2
1.5.1 Major System Component: Database . . . . .	2
1.5.2 Major System Component: User Interface . . . . .	2
1.6 Systems Goals . . . . .	2
1.7 System Overview and Diagram . . . . .	2
1.8 Technologies Overview . . . . .	3
<b>2 User Stories, Requirements, and Product Backlog</b>	<b>5</b>
2.1 Overview . . . . .	5
2.2 User Stories . . . . .	5
2.2.1 User Story #1 . . . . .	6
2.2.2 User Story #2 . . . . .	6
2.2.3 User Story #3 . . . . .	6
2.2.4 User Story #4 . . . . .	7
2.2.5 User Story #5 . . . . .	7
2.2.6 User Story #6 . . . . .	7
2.2.7 User Story #7 . . . . .	7
2.2.8 User Story #8 . . . . .	7
2.2.9 User Story #9 . . . . .	7
2.2.10 User Story #10 . . . . .	8
2.2.11 User Story #11 . . . . .	8
2.2.12 User Story #12 . . . . .	8

2.2.13	User Story #13 . . . . .	8
2.2.14	User Story #14 . . . . .	9
2.2.15	User Story #15 . . . . .	9
2.2.16	User Story #16 . . . . .	9
2.3	Requirements and Design Constraints . . . . .	9
2.3.1	System Requirements . . . . .	9
2.3.2	Network Requirements . . . . .	10
2.3.3	Development Environment Requirements . . . . .	10
2.3.4	Project Management Methodology . . . . .	10
2.4	Product Backlog . . . . .	10
2.4.1	Sprint 1 Backlog . . . . .	10
2.4.2	Sprint 2 Backlog . . . . .	11
2.4.3	Sprint 3 Backlog . . . . .	11
2.4.4	Sprint 3.5 Backlog . . . . .	11
2.4.5	Sprint 4 Backlog . . . . .	12
2.4.6	Sprint 5 Backlog . . . . .	12
2.4.7	Sprint 5 Fixed Bug Backlog . . . . .	13
2.4.8	Sprint 6 Backlog . . . . .	13
2.5	Research or Proof of Concept Results . . . . .	13
<b>3</b>	<b>Project Overview</b> . . . . .	<b>15</b>
3.1	Team Members and Roles . . . . .	15
3.2	Project Management Approach . . . . .	15
3.3	Stakeholder Information . . . . .	16
3.3.1	Customer or End User (Product Owner) . . . . .	16
3.3.2	Management or Instructor (Scrum Master) . . . . .	16
3.3.3	Developers –Testers . . . . .	16
3.4	Budget . . . . .	16
3.5	Intellectual Property and Licensing . . . . .	17
3.6	Sprint Overview . . . . .	17
3.6.1	Sprint 1: Initial User Story and Requirements Gathering . . . . .	17
3.6.2	Sprint 2: Database Creation and Starting Pages . . . . .	17
3.6.3	Sprint 3: Functionality Creation . . . . .	18
3.6.4	Sprint 4: Development Updates and Payroll/Billing . . . . .	18
3.6.5	Sprint 5: Updates Bug Fixes and Functionality . . . . .	20
3.6.6	Sprint 6: Updates, Fixes, and Iteration Packaging . . . . .	21
3.7	Terminology and Acronyms . . . . .	21
3.8	Sprint Schedule . . . . .	21
3.9	Backlogs . . . . .	22
3.9.1	Sprint 1 Backlog . . . . .	22
3.9.2	Sprint 2 Backlog . . . . .	22
3.9.3	Sprint 3 Backlog . . . . .	22
3.9.4	Sprint 3.5 Backlog . . . . .	23
3.9.5	Sprint 4 Backlog . . . . .	23
3.9.6	Sprint 5 Backlog . . . . .	23
3.9.7	Sprint 5 Fixed Bug Backlog . . . . .	24
3.9.8	Sprint 6 Backlog . . . . .	24
3.10	Development Environment . . . . .	25
3.10.1	Source Code Development . . . . .	25
3.10.2	MySQL Database . . . . .	25
3.10.3	PyQt . . . . .	25
3.11	Development IDE and Tools . . . . .	25
3.12	Source Control . . . . .	26
3.13	Dependencies . . . . .	26
3.14	Build Environment . . . . .	26

3.15	Development Machine Setup	26
<b>4</b>	<b>Design and Implementation</b>	<b>27</b>
4.1	Architecture and System Design	27
4.1.1	Design Selection	28
4.1.2	Data Structures and Algorithms	28
4.1.3	Data Flow	28
4.1.4	Communications	28
4.1.5	Classes	28
4.1.6	UML	28
4.1.7	GUI	28
4.1.8	MVVM, etc	28
4.2	Major Component #1	28
4.2.1	Technologies Used	28
4.2.2	Component Overview	28
4.2.3	Phase Overview	28
4.2.4	Architecture Diagram	28
4.2.5	Data Flow Diagram	28
4.2.6	Design Details	28
4.3	Major Component #2	29
4.3.1	Technologies Used	29
4.3.2	Component Overview	29
4.3.3	Phase Overview	29
4.3.4	Architecture Diagram	29
4.3.5	Data Flow Diagram	29
4.3.6	Design Details	29
4.4	Major Component #3	29
4.4.1	Technologies Used	29
4.4.2	Component Overview	30
4.4.3	Phase Overview	30
4.4.4	Architecture Diagram	30
4.4.5	Data Flow Diagram	30
4.4.6	Design Details	30
<b>5</b>	<b>System and Unit Testing</b>	<b>31</b>
5.1	Overview	31
5.2	Dependencies	31
5.3	Test Setup and Execution	31
5.4	System Testing	32
5.5	System Integration Analysis	34
5.6	Risk Analysis	34
5.6.1	Risk Mitigation	34
5.7	Successes, Issues and Problems	35
5.7.1	Changes to the Backlog	35
<b>6</b>	<b>Prototypes</b>	<b>37</b>
6.1	Sprint 1 Prototype	37
6.1.1	Deliverable	38
6.1.2	Backlog	38
6.1.3	Success/Fail	38
6.2	Sprint 2 Prototype	38
6.2.1	Deliverable	39
6.2.2	Backlog	39
6.2.3	Success/Fail	39
6.3	Sprint 3 Prototype	40
6.3.1	Deliverable	40

6.3.2	Backlog . . . . .	41
6.3.3	Success/Fail . . . . .	41
6.4	Sprint 3.5 and 4 Prototype . . . . .	41
6.4.1	Deliverable . . . . .	44
6.4.2	Backlog . . . . .	44
6.4.3	Success/Fail . . . . .	45
6.5	Sprint 5 Prototype . . . . .	45
6.5.1	Deliverable . . . . .	47
6.5.2	Backlog . . . . .	47
6.5.3	Success/Fail . . . . .	48
<b>7</b>	<b>Release – Setup – Deployment</b>	<b>49</b>
7.1	Deployment Information and Dependencies . . . . .	49
7.2	Setup Information . . . . .	50
7.2.1	Installing Python 3 . . . . .	50
7.2.2	Installing PyQt 4 and Designer toolkit . . . . .	50
7.2.3	Installing MySQL . . . . .	50
7.2.4	Running Project . . . . .	50
7.3	System Version Information . . . . .	51
7.4	Future Work . . . . .	51
<b>8</b>	<b>User Documentation</b>	<b>53</b>
8.1	User Guide . . . . .	53
8.1.1	Entering the Software . . . . .	53
8.1.2	Admin Landing . . . . .	53
8.1.3	Teacher Landing . . . . .	68
8.2	Installation Guide . . . . .	75
<b>9</b>	<b>Class Index</b>	<b>77</b>
<b>10</b>	<b>Class Documentation</b>	<b>79</b>
10.1	Add Class Reference . . . . .	79
10.1.1	Constructor & Destructor Documentation . . . . .	79
10.1.2	Member Function Documentation . . . . .	79
10.2	Add admin Reference . . . . .	80
10.2.1	Constructor & Destructor Documentation . . . . .	80
10.2.2	Member Function Documentation . . . . .	80
10.3	Add new teacher Reference . . . . .	80
10.3.1	Constructor & Destructor Documentation . . . . .	80
10.3.2	Member Function Documentation . . . . .	80
10.4	addLocation Reference . . . . .	81
10.4.1	Constructor & Destructor Documentation . . . . .	81
10.4.2	Member Function Documentation . . . . .	81
10.5	Addstu details Reference . . . . .	81
10.5.1	Constructor & Destructor Documentation . . . . .	82
10.5.2	Member Function Documentation . . . . .	82
10.6	Addstu window Reference . . . . .	82
10.6.1	Constructor & Destructor Documentation . . . . .	82
10.6.2	Member Function Documentation . . . . .	82
10.7	Addstu window admin Reference . . . . .	83
10.7.1	Constructor & Destructor Documentation . . . . .	83
10.7.2	Member Function Documentation . . . . .	83
10.8	Admin Reference . . . . .	84
10.8.1	Constructor & Destructor Documentation . . . . .	84
10.8.2	Member Function Documentation . . . . .	84
10.8.3	Member Function continued Documentation . . . . .	87

10.9 Admin info Reference . . . . .	87
10.9.1 Constructor & Destructor Documentation . . . . .	88
10.10 Admin list Reference . . . . .	88
10.10.1 Constructor & Destructor Documentation . . . . .	88
10.10.2 Member Function Documentation . . . . .	88
10.11 Adv class Reference . . . . .	89
10.11.1 Constructor & Destructor Documentation . . . . .	89
10.11.2 Member Function Documentation . . . . .	89
10.12 Adv student Reference . . . . .	89
10.12.1 Constructor & Destructor Documentation . . . . .	89
10.12.2 Member Function Documentation . . . . .	89
10.13 Adv teacher Reference . . . . .	89
10.13.1 Constructor & Destructor Documentation . . . . .	90
10.13.2 Member Function Documentation . . . . .	90
10.14 Assign class Reference . . . . .	90
10.14.1 Constructor & Destructor Documentation . . . . .	90
10.14.2 Member Function Documentation . . . . .	90
10.15 Billing history Reference . . . . .	91
10.15.1 Constructor & Destructor Documentation . . . . .	91
10.15.2 Member Function Documentation . . . . .	91
10.16 Print Hist Reference . . . . .	91
10.16.1 Constructor & Destructor Documentation . . . . .	91
10.16.2 Member Function Documentation . . . . .	92
10.17 Change PW Reference . . . . .	92
10.17.1 Constructor & Destructor Documentation . . . . .	92
10.17.2 Member Function Documentation . . . . .	92
10.18 Change user Reference . . . . .	92
10.18.1 Constructor & Destructor Documentation . . . . .	93
10.18.2 Member Function Documentation . . . . .	93
10.19 Class list Reference . . . . .	93
10.19.1 Constructor & Destructor Documentation . . . . .	93
10.19.2 Member Function Documentation . . . . .	93
10.20 Class reg Reference . . . . .	94
10.20.1 Constructor & Destructor Documentation . . . . .	94
10.20.2 Member Function Documentation . . . . .	94
10.21 Credits Reference . . . . .	94
10.21.1 Constructor & Destructor Documentation . . . . .	95
10.21.2 Member Function Documentation . . . . .	95
10.22 Full pay Reference . . . . .	95
10.22.1 Constructor & Destructor Documentation . . . . .	95
10.22.2 Member Function Documentation . . . . .	95
10.23 Enter hours Reference . . . . .	96
10.23.1 Constructor & Destructor Documentation . . . . .	96
10.23.2 Member Function Documentation . . . . .	96
10.24 Print fees Reference . . . . .	97
10.24.1 Constructor & Destructor Documentation . . . . .	97
10.24.2 Member Function Documentation . . . . .	97
10.25 fees Reference . . . . .	97
10.25.1 Constructor & Destructor Documentation . . . . .	97
10.25.2 Member Function Documentation . . . . .	98
10.26 login Reference . . . . .	98
10.26.1 Constructor & Destructor Documentation . . . . .	98
10.26.2 Member Function Documentation . . . . .	99
10.27 My info Reference . . . . .	99
10.27.1 Constructor & Destructor Documentation . . . . .	99

10.27.2 Member Function Documentation . . . . .	99
10.28 Navi Reference . . . . .	100
10.28.1 Constructor & Destructor Documentation . . . . .	100
10.28.2 Member Function Documentation . . . . .	100
10.29 Partial pay Reference . . . . .	100
10.29.1 Constructor & Destructor Documentation . . . . .	100
10.29.2 Member Function Documentation . . . . .	100
10.30 Part pay dialog Reference . . . . .	101
10.30.1 Constructor & Destructor Documentation . . . . .	101
10.30.2 Member Function Documentation . . . . .	101
10.31 refund Reference . . . . .	101
10.31.1 Constructor & Destructor Documentation . . . . .	102
10.31.2 Member Function Documentation . . . . .	102
10.32 removeAdmin Reference . . . . .	102
10.32.1 Constructor & Destructor Documentation . . . . .	102
10.32.2 Member Function Documentation . . . . .	102
10.33 removeClass Reference . . . . .	103
10.33.1 Constructor & Destructor Documentation . . . . .	103
10.33.2 Member Function Documentation . . . . .	103
10.34 removeStudentData Reference . . . . .	103
10.34.1 Constructor & Destructor Documentation . . . . .	103
10.34.2 Member Function Documentation . . . . .	103
10.35 removeTeacher Reference . . . . .	104
10.35.1 Constructor & Destructor Documentation . . . . .	104
10.35.2 Member Function Documentation . . . . .	104
10.36 Roll Reference . . . . .	104
10.36.1 Constructor & Destructor Documentation . . . . .	104
10.36.2 Member Function Documentation . . . . .	104
10.37 Print Roll Reference . . . . .	105
10.37.1 Constructor & Destructor Documentation . . . . .	105
10.37.2 Member Function Documentation . . . . .	105
10.38 schedule print Reference . . . . .	105
10.38.1 Constructor & Destructor Documentation . . . . .	106
10.38.2 Member Function Documentation . . . . .	106
10.39 Search class Reference . . . . .	106
10.39.1 Constructor & Destructor Documentation . . . . .	106
10.39.2 Member Function Documentation . . . . .	106
10.40 Search teacher Reference . . . . .	107
10.40.1 Constructor & Destructor Documentation . . . . .	107
10.40.2 Member Function Documentation . . . . .	108
10.41 Search student Reference . . . . .	108
10.41.1 Constructor & Destructor Documentation . . . . .	109
10.41.2 Member Function Documentation . . . . .	109
10.42 Update teacher Reference . . . . .	109
10.42.1 Constructor & Destructor Documentation . . . . .	110
10.42.2 Member Function Documentation . . . . .	110
10.43 Set semester Reference . . . . .	110
10.43.1 Constructor & Destructor Documentation . . . . .	110
10.43.2 Member Function Documentation . . . . .	111
10.44 Show hours Reference . . . . .	111
10.44.1 Constructor & Destructor Documentation . . . . .	111
10.44.2 Member Function Documentation . . . . .	111
10.45 Stu add Reference . . . . .	111
10.45.1 Constructor & Destructor Documentation . . . . .	112
10.45.2 Member Function Documentation . . . . .	112



10.46	Stu reg Reference . . . . .	112
10.46.1	Constructor & Destructor Documentation . . . . .	112
10.46.2	Member Function Documentation . . . . .	112
10.47	Stu pay Reference . . . . .	112
10.47.1	Constructor & Destructor Documentation . . . . .	113
10.47.2	Member Function Documentation . . . . .	113
10.48	Stu reg window Reference . . . . .	113
10.48.1	Constructor & Destructor Documentation . . . . .	114
10.48.2	Member Function Documentation . . . . .	114
10.49	Student schedule Reference . . . . .	114
10.49.1	Constructor & Destructor Documentation . . . . .	114
10.49.2	Member Function Documentation . . . . .	115
10.50	Teacher Reference . . . . .	115
10.50.1	Constructor & Destructor Documentation . . . . .	115
10.50.2	Member Function Documentation . . . . .	115
10.51	Teacher history dialog Reference . . . . .	117
10.51.1	Constructor & Destructor Documentation . . . . .	117
10.51.2	Member Function Documentation . . . . .	117
10.52	Teacher history window Reference . . . . .	117
10.52.1	Constructor & Destructor Documentation . . . . .	117
10.52.2	Member Function Documentation . . . . .	117
10.53	Teacher payrate dialog Reference . . . . .	118
10.53.1	Constructor & Destructor Documentation . . . . .	118
10.53.2	Member Function Documentation . . . . .	118
10.54	Teacher payrate Reference . . . . .	118
10.54.1	Constructor & Destructor Documentation . . . . .	118
10.54.2	Member Function Documentation . . . . .	119
10.55	Teacher schedule Reference . . . . .	119
10.55.1	Constructor & Destructor Documentation . . . . .	119
10.55.2	Member Function Documentation . . . . .	119
10.56	Tuition Reference . . . . .	120
10.56.1	Constructor & Destructor Documentation . . . . .	120
10.56.2	Member Function Documentation . . . . .	120
10.57	Update fees Reference . . . . .	121
10.57.1	Constructor & Destructor Documentation . . . . .	121
10.57.2	Member Function Documentation . . . . .	121
10.58	Update rates Reference . . . . .	122
10.58.1	Constructor & Destructor Documentation . . . . .	122
10.58.2	Member Function Documentation . . . . .	122
<b>Bibliography</b>		<b>125</b>
<b>Software Agreement</b>		<b>SA-1</b>
<b>A Product Description</b>		<b>A-1</b>
<b>B Sprint Reports</b>		<b>B-1</b>
1	Sprint Report #1 . . . . .	B-1
1.1	Team Members: . . . . .	B-1
1.2	Customer description . . . . .	B-1
1.3	Overview of the project: . . . . .	B-1
1.4	Project Environment: . . . . .	B-2
1.5	Project deliverables of Sprint 1: . . . . .	B-2
1.6	User Stories . . . . .	B-2
1.7	Product Backlog: . . . . .	B-3
1.8	Research . . . . .	B-4

	1.9	Final Framework Decision . . . . .	B-5
	1.10	Foreseeable issues . . . . .	B-5
2		Sprint Report #2 . . . . .	B-5
	2.1	Team Members: . . . . .	B-5
	2.2	Prototype Progress . . . . .	B-6
	2.3	Project deliverables of Sprint 2: . . . . .	B-6
	2.4	Database Creation . . . . .	B-6
	2.5	GUI Work . . . . .	B-6
	2.6	Sprint 2 Issues . . . . .	B-8
	2.7	Client Interactions . . . . .	B-8
	2.8	Group Meeting . . . . .	B-8
	2.9	Work Distribution . . . . .	B-9
3		Sprint Report #3 . . . . .	B-9
	3.1	Team Members: . . . . .	B-9
	3.2	Prototype Progress . . . . .	B-9
	3.3	Project deliverables of Sprint 3: . . . . .	B-9
	3.4	Search Pages . . . . .	B-10
	3.5	Add A Class . . . . .	B-10
	3.6	Role Sheet . . . . .	B-10
	3.7	Update Pages . . . . .	B-11
	3.8	Assign Teacher to a Class . . . . .	B-11
	3.9	Sprint 3 Issues . . . . .	B-11
	3.10	Client Interactions . . . . .	B-11
	3.11	Group Meeting . . . . .	B-11
	3.12	Work Distribution . . . . .	B-11
4		Sprint Report #4 . . . . .	B-12
	4.1	Team Members: . . . . .	B-12
	4.2	Prototype Progress . . . . .	B-12
	4.3	Project deliverables of Sprint 3.5 and 4: . . . . .	B-12
	4.4	Sprint 3.5 . . . . .	B-13
	4.5	Prorated Refunds . . . . .	B-13
	4.6	Enter Staff Hours . . . . .	B-13
	4.7	Enter Teacher Wages . . . . .	B-14
	4.8	Enter Tuition Rates and Fees . . . . .	B-14
	4.9	Teacher History . . . . .	B-14
	4.10	Set Semester . . . . .	B-14
	4.11	partial payment . . . . .	B-15
	4.12	Student's owe . . . . .	B-15
	4.13	enter payment . . . . .	B-15
	4.14	bill history . . . . .	B-16
	4.15	Sprint 3.5 and 4 Issues . . . . .	B-16
	4.16	Client Interactions . . . . .	B-17
	4.17	Group Meeting . . . . .	B-17
	4.18	Work Distribution . . . . .	B-17
5		Sprint Report #5 . . . . .	B-18
	5.1	Team Members: . . . . .	B-18
	5.2	Prototype Progress . . . . .	B-18
	5.3	Project deliverables of Sprint 5: . . . . .	B-18
	5.4	Sprint 5 . . . . .	B-19
	5.5	Prorated Refunds . . . . .	B-19
	5.6	Enter Staff Hours . . . . .	B-19
	5.7	Enter Teacher Wages . . . . .	B-20
	5.8	Enter Tuition Rates and Fees . . . . .	B-20
	5.9	Bug Fixes . . . . .	B-20
	5.10	Updates Crossover . . . . .	B-20

5.11	Approved/Rejected Student Updates . . . . .	B-20
5.12	Admin List . . . . .	B-21
5.13	Add/Remove Locations . . . . .	B-21
5.14	Removal Functions . . . . .	B-21
5.15	Sprint 5 Issues . . . . .	B-22
5.16	Client Interactions . . . . .	B-22
5.17	Group Meeting . . . . .	B-22
5.18	Work Distribution . . . . .	B-22
<b>C</b>	<b>Industrial Experience and Resumes</b>	<b>C-1</b>
1	Resumes . . . . .	C-1
2	Industrial Experience Reports . . . . .	C-4
2.1	Marcus Berger . . . . .	C-4
2.2	Dicheng Wu . . . . .	C-4
<b>D</b>	<b>Acknowledgment</b>	<b>D-1</b>
<b>E</b>	<b>Supporting Materials</b>	<b>E-1</b>



---

## List of Figures

---

1.1	Main GUI Overview . . . . .	3
1.2	Main Database Overview . . . . .	4
3.1	Add Class Form . . . . .	19
6.1	Tables currently in database . . . . .	38
6.2	Current iteration of the log in page . . . . .	39
8.1	Log in page . . . . .	54
8.2	Admin Main Page . . . . .	54
8.3	Teacher Search Window . . . . .	55
8.4	Update Teacher Form . . . . .	56
8.5	The Admin Hours Page . . . . .	57
8.6	Assign Class Window . . . . .	58
8.7	Teacher History . . . . .	59
8.8	Class Search Window . . . . .	60
8.9	Set Semester Window . . . . .	61
8.10	Student Search Window . . . . .	62
8.11	Student Credit Window . . . . .	63
8.12	First Student Registration . . . . .	64
8.13	Current Student Registration Window . . . . .	65
8.14	Add New Student Window . . . . .	66
8.15	Partial Payment Window . . . . .	67
8.16	Enter Pay Rate Window . . . . .	68
8.17	Student Balance Window . . . . .	69
8.18	Student Billing History . . . . .	70
8.19	Tuition and Fees . . . . .	70
8.20	Student Search Window . . . . .	71
8.21	Student Schedule Window . . . . .	72
8.22	Teacher Schedule . . . . .	73
8.23	Class Role Sheet . . . . .	74
8.24	Python zip files . . . . .	75
B.1	DanceSoft Trello Board . . . . .	B-4
B.2	Tables currently in database . . . . .	B-7
B.3	Current iteration of the log in page . . . . .	B-7
B.4	Current iteration of the Admin Landing page . . . . .	B-8
B.5	Search Pages . . . . .	B-10



---

## List of Tables

---





---

## List of Algorithms

---

1	Calculate $y = x^n$ . . . . .	27
---	-------------------------------	----



---

## Overview Statements

---

### 0.1 Mission Statement

The mission of the DanceSoft team is to create a efficient and effective system for the Academy of Dance Arts. So data can be managed clearly and easily to manage the academy's day to day needs. [MB]

### 0.2 Elevator Pitch

The DanceSoft project is a data management software for the Academy of Dance Arts in Rapid City. The project aims to produce a simple and more effective data management tool then what is currently in use at the academy. This will be accomplished through the use of a database and a simple user friendly interface, which will allow faculty and students to easily accomplish their needs, whether that's registering for a class, getting a role sheet or just general people management. DanceSoft will provide effective management tools so people spend less time at their computer, and more time dancing. [MB]



---

## Document Preparation and Updates

---

Current Version [X.X.X]

*Prepared By:*  
*Marcus Berger #1*  
*Dicheng Wu #2*

***Revision History***

<b>Date</b>	<b>Author</b>	<b>Version</b>	<b>Comments</b>
<b>9/12/15</b>	Marcus Berger and Dicheng Wu	1.0.0	Initial version
<b>10/8/15</b>	Marcus Berger	1.0.1	Sprint Report 1
<b>10/22/15</b>	Marcus Berger	1.0.1	Initial Overview
<b>10/23/15</b>	Marcus Berger	1.0.1	Initial Mission Statement and project.tex
<b>10/28/15</b>	Marcus Berger	1.0.1	Initial Requirements, Testing, and develop.tex
<b>11/5/15</b>	Marcus Berger and Dicheng Wu	1.0.2	Sprint Report 2
<b>11/5/15</b>	Marcus Berger	1.0.2	Updated Overview
<b>12/4/15</b>	Marcus Berger and Dicheng Wu	1.0.3	Sprint Report 3
<b>12/8/15</b>	Marcus Berger	1.0.3	Updated Project, Uploaded Resume, and Add Acknowledgments
<b>12/9/15</b>	Marcus Berger	1.0.3	Updated Requirements (future user stories marked with update time frame)
<b>12/9/15</b>	Dicheng Wu	1.0.3	Uploaded Resume
<b>12/9/15</b>	Marcus Berger	1.0.3	Minor Update to Testing
<b>12/10/15</b>	Dicheng Wu	1.0.3	Updates to Design and Develop
<b>12/10/15</b>	Marcus Berger	1.0.3	Product Contract Description
<b>3/25/16</b>	Marcus Berger	1.1.0	Merged over to new design template
<b>3/30/16</b>	Marcus Berger	1.1.1	Reiterate the overview section and fill out the experience and bibliography
<b>4/5/16</b>	Marcus Berger	1.1.2	Added new software contract
<b>4/12/16</b>	Marcus Berger	1.1.2	Rewrote requirements section based on new requirements
<b>4/15/16</b>	Marcus Berger	1.1.2	Finished the overview section
<b>4/25/16</b>	Marcus Berger	1.1.2	Started user guide section
<b>4/26/16</b>	Marcus Berger	1.1.2	Finished user guide section
<b>4/26/16</b>	Dicheng Wu	1.1.2	Started Class Documentation
<b>4/26/16</b>	Marcus Berger	1.1.2	Started writing the deploy section
<b>4/27/16</b>	Marcus Berger	1.1.2	Finished the deploy section and wrote short industrial experience summaries for the team
<b>4/27/16</b>	Marcus Berger	1.1.2	Added the product description
<b>4/28/16</b>	Marcus Berger	1.1.2	Added the prototypes section
<b>4/28/16</b>	Dicheng Wu	1.1.2	Added the class documentation

# 1

---

## Overview and concept of operations

---

This chapter contains a general overview of the DanceSoft project. [MB]

### 1.1 Team Members and Team Name

The DanceSoft Team consist of:

1. Marcus Berger
2. Dicheng Wu

### 1.2 Client

The stakeholder and sponsoring customer for this project is Dr. Jeff McGough a computer science professor at the South Dakota School of Mines and Technology and the vice president of the Academy of Dance Arts in Rapid City, South Dakota. Well not the sponsor of the project Dr. McGough's wife, Julie McFarland is also a key part of the customer base and a stakeholder as the owner and artistic director of the Academy of Dance Arts in Rapid City, South Dakota. The clients main goal is to eventually use the software after some iterations to manage the day to day activities of the Academy, and as the number of iterations of the project increase add student interfaces and refines the project to a point of business viability and security.

### 1.3 Project

This first iteration of the project is a proof of concept for the DanceSoft project for purposed use at the Rapid City Academy of Dance Arts. The project after this iteration is meant to be a minimum viable product to give the client as proof of project viability. After this project it will be up to the client, Jeff McGough, to decide whether to keep the project, look elsewhere, or hand over the project to a following senior design team.

As stated above the project is meant to be a minimum viable project, or proof of concept. The system general purpose is running the day to day administrative duties of the Academy of Dance Arts. These duties can range from, but are not necessarily limited to: enter student and class information, payments and billing, or managing employees information.

#### 1.3.1 Purpose of the System

The purpose of this system is to provide a system for the Academy of Dance Arts to manage their day to day operations, their employees, and their students. These day to day operations range from assigning teachers to classes, looking up information, printing roles sheets for classes, etc. Also the system must accomplish these task using a user friendly interface, that is as intuitive as possible. The project looks to accomplish this purpose through a local graphical interface and a back end mySQL database for the storage of the data.

## 1.4 Business Need

The customer needs the team to develop a software solution which can run the dance academy in an effective manner. The product also needs to handle changing classes from year to year without needing to be updated. This means that the software needs to sync with various information, and handle new information such as class rosters, prices, clothing requirements for classes, changes in the employment roster, and many other changes that can occur in the running of the dance school.

This project as a whole needs to be an improvement on the current system in use by the customer and provide an easy and efficient way to run the clients business. This project will accomplish the data manipulation task through the back-end MySQL database, and the ease of use will be handled with a simple PyQt interface. The academy also need the system to work with the employees of the academy which can be divided into admin and teacher categories within the system. The team accomplishes this through a log in system to different landing pages which contain the various functionality completed in this first iteration of the project, as laid out in the user stories listed in this document.

## 1.5 Deliverables

Listed below are the deliverable major system components for this project. [MB]

### 1.5.1 Major System Component: Database

The first major component of the system is the MySQL relational database. The database contains the Academy's data and is the core of the back-end side of the software. This database will be the conduit for most of the systems interactions with the data. The database will live within a local computer provided by the user for this first iteration of the software.

### 1.5.2 Major System Component: User Interface

The second major component of the system is the front-end user interface for admins and teachers. This will be the only part of the system most users ever see, and will provide an effective means to complete the desired user task. This is accomplished through the use of pages created in PyQt with interfaces to give users effective ways to interact with the database and the necessary data for the requested operations and functions.

## 1.6 Systems Goals

The system needs to provide a solution which can run the dance studio data and some day to day activities in an effective and secure manner. This includes allowing teachers to print role sheets, look at schedules, and manage their information. Students need to have the ability to see information pertinent to them such as registration and class requirements. Owners and admins need to be able to use the system to manage their employees, the academy's students and it classes, and other administrative duties such as billing. Lastly this system as a whole also needs to be an improvement on the current system in use by the customer and provide an easier and more efficient way to run the clients business.

Overall the system goal is to provide a environment where academy owners, teachers, and students can effectively manage their personal needs and requirements for academy participation and continued operations.

## 1.7 System Overview and Diagram

Users will access this application through a local GUI or a web application depending on their position within the system. The GUI application after client approval or more iterations should reach a final goal of being deployed on the local machines within the Rapid City Academy of Dance Arts. When a local GUI user connects the user will navigate through the various pages listed above to the desired functionality. Figure 1.1 shows a simplistic view of the GUI Architecture. There are two major components to this iteration the project, database, and admin/teacher interface. Each section is described in more functional detail above and in **section 4 Design**



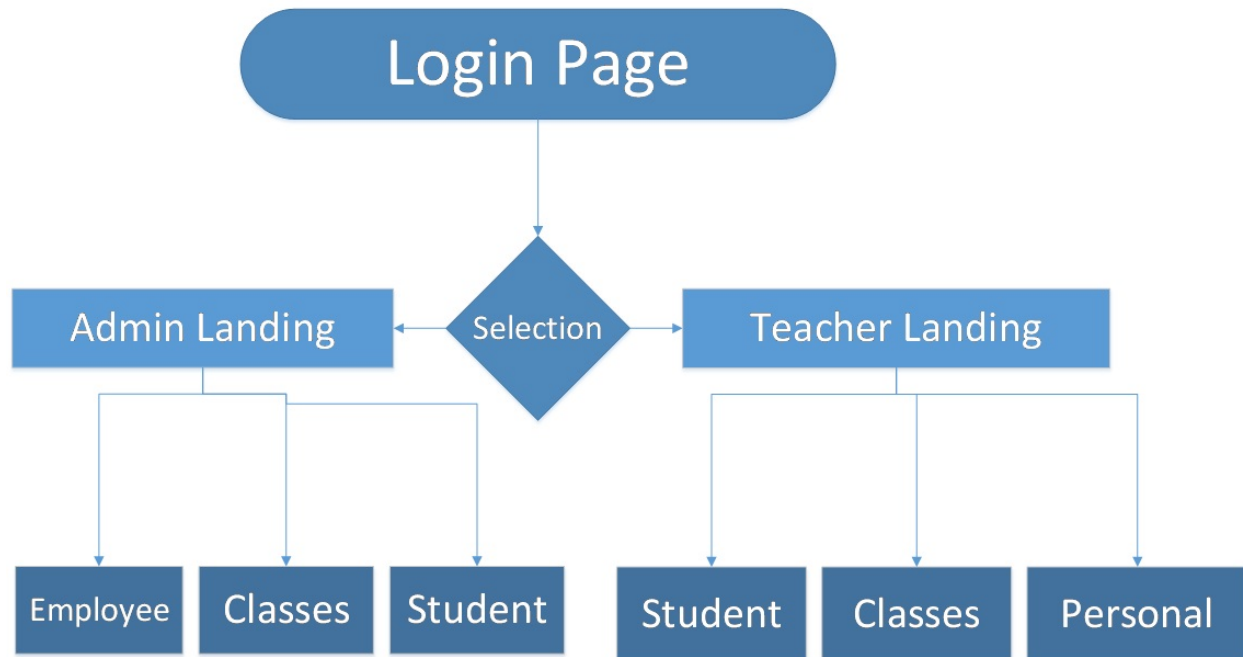


Figure 1.1: Main GUI Overview

**and Implementation** . The database architecture is a system of mySQL tables connected through the use of various keys. The tables correspond to the necessary information for various functionality within the system. [MB]

## 1.8 Technologies Overview

The primary Technologies for this projects are as follows:

1. Xcode and Visual Studio - Xcode, Microsoft Visual Studio 2015, and Python IDE were the primary development IDEs for this project. Of the three the ones the group used the most were Visual Studio 2015, and the Python IDE so the team could develop the project on the computers provided by the South Dakota School of Mines and Technology. Xcode test were done every so often to confirm the files worked cross-platform.
2. Python - the primary programming language for the project.
3. PyQt and Qt Designer - GUI package and development environment, these tools are publicly available and can be downloaded for free off the internet.
4. MySQL - MySQL provides the database and relational quires to manage the data and organize it within the system. This software is also free to use and can be downloaded from the MySQL website.

These technologies were selected after a first research sprint where research into programming language, GUI, and database options was conducted and the technologies were selected. A brief description of the research can be found in the sprint 1 report or the first prototype sections of this document. [MB]

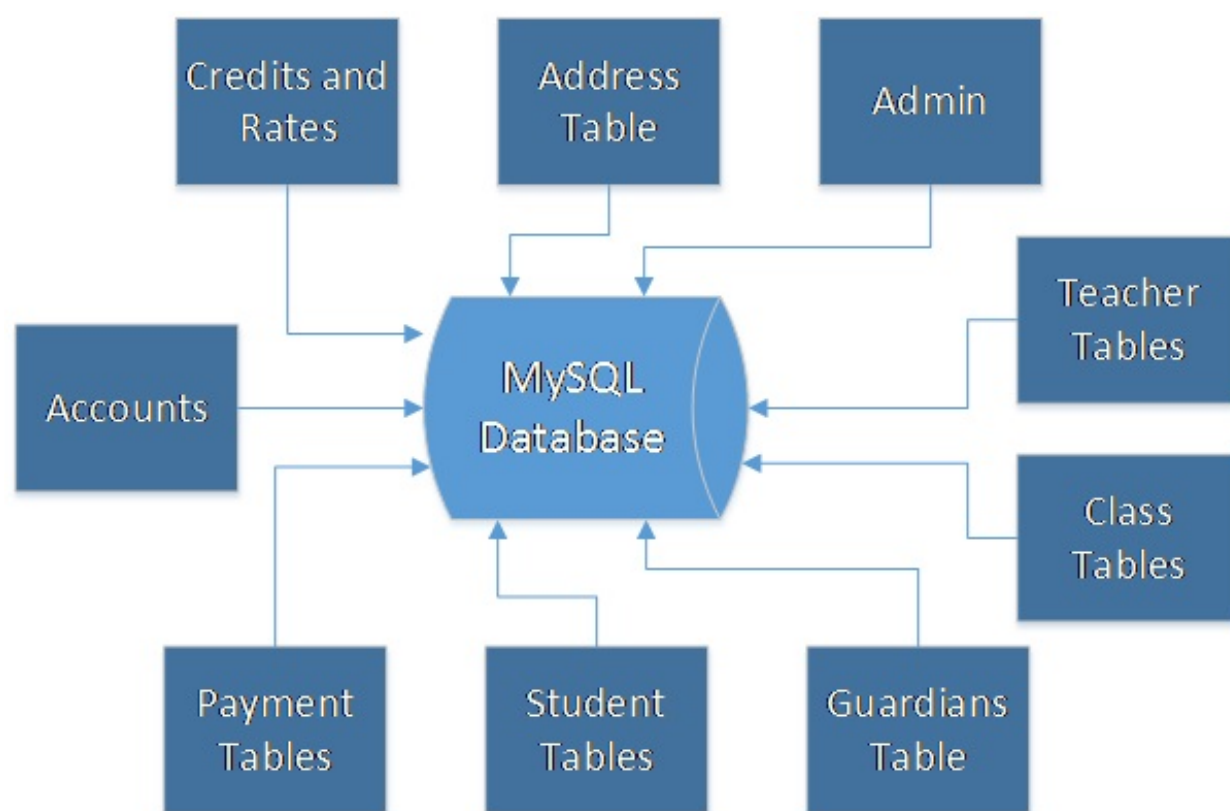


Figure 1.2: Main Database Overview

## 2

---

# User Stories, Requirements, and Product Backlog

---

## 2.1 Overview

This document covers the client information, an overview of the requirements of the system and the requirements laid out by the client. These requirements are laid out in this document with how the group plans to implement them within the system. [MB]

## 2.2 User Stories

After the requirement for the project were laid out the team created the user stories based on those requirement. The user stories the team came up with are as follows: [MB]

1. As a user i want to adjust students payment models
2. As the owner I would like to see automatic database backups.
3. As a student I would like to be able to register online (with special app). Classes must be approved before added.
4. As a student I would like to be able to search clothing requirements.
5. As a student I would like to know my billing.
6. As the owner I would like to indicate clothing requirements per class.
7. As a studio person, I would like to be able to add students to classes.
8. As a student, teacher etc, I would like to be able to look up a students class list.
9. As the teacher I would like to get a class role for each class.
10. Given a class list, I would like to get an invoice of the tuition due.
11. Studio would like to track payments and estimate remainder due. I would like to generate an invoice for this amount.
12. As a student I would like to be able to register online (with special app). Classes must be approved before added.
13. As a student I would like to know my billing.
14. As the owner I would like to track teacher hours and compute payroll.
15. As a studio employee I would like to open a registration pane and add student data
16. As the studio owner I would like to enter teacher information and look up information such as SS and pay rates.

17. As the owner, I would like to enter classes: time, location, registration cap. I would like to view this information later. I would like to assign instructors
18. As a user I want to have different payment models for different situations

### 2.2.1 User Story #1

As a user i want to adjust students payment models. This user stories means that the user should be able to go find a student, select that student and change the pay model to another existing payment model. Adding a payment model is part of a different user story.

#### 2.2.1.a User Story #1 Breakdown

Further breakdown for this user story: this functionally was further broken down into, allowing the student payment to be accept in many ways. Ways to make payment include credit, cash, check, and other. During the first iteration of this project this user story was partially completed. The user is able to process a students payment type and log the payments in a payment history.

#### 2.2.1.b User Story #1 Remaining

This user story did not reach full completion in this iteration of the project. The discount types that the academy uses exist within the database, however due to issues with the project the current iteration never reached the point of handling the payment models with in the interface. If next iteration proceeds directly from this iteration the following still need to be implemented:

1. handle the payment model GUI page
2. check to see if a student is on a certain payment model
3. handle discount changes in the database

### 2.2.2 User Story #2

As the owner I would like to see automatic database backups. This one is fairly simple the system will need to back up the data from the database locally or by an external provider.

#### 2.2.2.a User Story #2 Remaining

The DanceSoft team during this iteration of the project did not reach this user story before the submission deadline. The database will most likely be backed up on a machine provided by the client in the next iteration as the project becomes more refined and closer to business deployment.

### 2.2.3 User Story #3

As a student I would like to be able to register online (with special app). Classes must be approved before added. The special app references in this user story is the php student web interface that the team will create. The students will be able to go to the website, log in and register for classes along with other features listed in other user stories.

#### 2.2.3.a User Story #3 Breakdown

Also listed in this user story is the ability for admins to approve any class registrations by students before the registration is finalizes and submitted to the data base. This approval system needs to have three states. First is pending which will be the unanswered request in the system. Second is the approved option which will finalize the students registration and place them in their desired class. Lastly is denied which will not put the student in the class.

The team has created a version of this registration approval function, where a admin in the system can approve, or reject a students request to register. Also in this interface the admin can choose to refund the

student if they are dropping the class or just strictly remove the student if they don't need a refund or the class has not yet started.

After the first three sprints the student interface was dropped from this iteration of the project. Future iterations should be able to integrate the student web portal with this system due to the fact that the initial design for this project included considerations for what would be needed in the student interface.

### 2.2.3.b Further Breakdown

This perceptive interface should also include the options to manage the student information handled by the system. This include the registration functionality, views, updates, registrations, and submits. The interface will run the requirements of the student and their guardians, most of which are covered by other user requirements.

### 2.2.4 User Story #4

As a student I would like to know my billing. This should allow the users of the local interface to see what a student still owes and print a log of this information.

In the next project iteration this functionally should also cross over to the student interface. Within the student interface a given student should be able to see things like what they have left to pay, how much they have paid, when the next payment is due, and other pertinent information to the students billing.

### 2.2.5 User Story #5

As the owner I would like to indicate clothing requirements per class. The owner or other admin will be able to add clothing requirements to a specific class and change them in a class menu using an update form. This is accomplished in the system through a add class form where the admin can create a new class and indicate clothing requirements for that class. Also if the user wants to update clothing requirement they can use a class search feature to find and update the necessary requirements.

### 2.2.6 User Story #6

As a studio person, I would like to be able to add students to classes. This options will allow all employees to request a specific student be added to a class. This will sent a request to an admin which will need to approve the request like a normal registration or approval by teachers.

In a future iteration something akin to this functionally will also need to be added to the future student interface to allow student to register through the online portal.

### 2.2.7 User Story #7

As a teacher or admin I would like to be able to look up a students class list. Users need to select a student and see what their class schedule is and which class they have pending registrations for. This will be accomplished through an output dialog that pops up to display the students schedule and the pending registration interface.

### 2.2.8 User Story #8

As the teacher I would like to get a class role for each class. Users need to be able to select a class and see who is enrolled in it. Also the list needs to be printable so teachers can take role at a class. This is done through a window where the teacher can select one of the classes they are teaching and print the role sheet for that class.

### 2.2.9 User Story #9

Given a class list, I would like to get an invoice of the tuition due. Users should be able to get an invoice for their billing based on the number of classes being taken, and the payment model the student is placed under.

### **2.2.9.a User Story #9 breakdown**

Currently this is accomplished through using the tuition rates logged in the database by the user, which were created using the academy payment models on their website. The hours a student is enrolled in are calculated to generate an invoice.

### **2.2.9.b User Story #9 remaining**

This interface will need to be expanded and adapted as the next iteration modifies or changes the interface structure or mechanics.

### **2.2.10 User Story #10**

Studio would like to track payments and estimate remainder due. I would like to generate an invoice for this amount. This user story follows user story 10. The system should track payments made and given those payments calculate what students or parents have left to pay.

Based off of the amounts calculated by the interface in user story 10 the student can submit a payment and the employee of the academy can see and print remaining due.

### **2.2.10.a User Story #10 remaining**

This interface will also need to be expanded and adapted as the next iteration modifies or changes the interface structure or mechanics.

### **2.2.11 User Story #11**

As the owner I would like to track teacher hours and compute payroll. Hours for teachers will need to be approved by an admin within their interface. Also calculations will be made based on that teachers pay rate to compute their pay. Lastly tax algorithms will need to be used to effectively make sure that tax are withdrawn correctly and neither the teacher or the academy will be liable in the case of audits.

### **2.2.11.a User Story #11 breakdown**

The team was able to create different pay names and pay rates for each individual employee to generate a wage for the teacher. The teacher will then be able to see the hours logged. The admin can also change the pay rates for different pay names such as driving or off-site rates.

Teachers and employees are also given a way to submit hours through their interface that can then be viewed by the owner.

### **2.2.11.b User Story #11 remaining**

During the first iteration of the team was able to complete the framework for generating an employee's gross wage. The tax calculations will need to be added and adapted in a future iteration of this project.

### **2.2.12 User Story #12**

As a studio employee I would like to open a registration pane and add student data. The employees of the academy should be able to modify student registration information. This will be used should the student's information change, or if the student's information was entered incorrectly.

This is done through an update form where the user can modify the information and submit the updates to the database.

### **2.2.13 User Story #13**

As the studio owner I would like to enter teacher information and look up information such as address and birth date. The system will provide the owner with the ability to search, view, and modify information within the system.

### 2.2.13.a User Story #13 Breakdown

Search and view functions exist for all level of users with different results in different areas. Examples being students need to see class information, teachers should search classes and students, admin should be able to see all information.

### 2.2.13.b User Story #13 Remaining

In future iterations this functionality will also be added and adapted for the future student web portal.

### 2.2.14 User Story #14

As the owner, I would like to enter classes: time, location, registration cap. I would like to view this information later. I would like to assign instructors. Simply put this is the ability of the owner to add a class to the academy's roster. Which is done through an add information form which checks required information and submits it to the database.

This information can later be viewed and updated through a class search and update function. While assign teacher to classes is done through a separate list view function

### 2.2.15 User Story #15

As a user I want to have different payment models for different situations. Allow the owner the ability to change, add, and select different payment methods for billing based on a number of factors. These factors include time of registration, dropped classes, admin selects payment options for a specific situation, etc.

### 2.2.15.a User Story #15 Remaining

The team during this iteration of the project was able to create the framework in the database necessary for this user story, however the team was unable to impliment this functionality in the interface during this iteration.

### 2.2.16 User Story #16

As a admin I want to be able to set level of permissions with information.

This user story is the basis for the log in system where the user can create a teacher, with default level permission. Then add the user as an admin if they choose to. The user can also remove admin complete from the system or just that employees admin credentials.

## 2.3 Requirements and Design Constraints

This section discusses what requirements exist that deal with meeting the business needs the customer has. For the DanceSoft these include system needs to run the academy, network connection issues, and some environment requirements laid out by both the client and the senior design requirements. [MB]

### 2.3.1 System Requirements

The system requirements laid out by the clients are the necessary features laid out in the user stories above. There was no preference on language or GUI environment on the part of the client. Due to some of the user stories and information handled within the system a level of security becomes a system requirement.

These requirements will carry over in general to any future iterations of this project as other iterations continue to refine, adapt, and rework the various functions of the project.

### 2.3.2 Network Requirements

The network inside the Academy has connection issues and therefore a cloud or online based data storage option is not a highly advised possibility. The network issues within the school means the system will be contained in a local system to provide more stability within the system.

When the student interface is integrated into the overall projects the network requirement may change or need to be adapted through iterations.

### 2.3.3 Development Environment Requirements

The academy runs on a Mac currently so the system must work on the Mac OSX operating system upon completion. While not required the project is developed in Python, so the end product will work cross platform. So that if the academy ever switches operating systems or if the academy is ever sold the system will work if the new owners run windows. If they choose to use it. Currently the client will store this iteration of the project in a local Linux box, which the client will provide to the team at the time of data transfer.

### 2.3.4 Project Management Methodology

The client requires a weekly meeting every Wednesday at 2:00 p.m. to check on the progress of the system. These meetings vary on topic and length depending on the needs of the project and the status of task. The senior design class requires that this project be in an acceptable iteration in six sprints that are all roughly three weeks long, with a week long results period after each one. Another project requirement is the presentations that are required by the senior design class. These presentations occur twice every semester usually after the first and last sprint each semester. Each presentation covers the content of the project up to that point, and updates on topics such as risk mitigation, budget, and current prototypes. Lastly it was requested by Dr. McGough as part of senior design and as the client that we provide him with access to the Github repository for the project and the Trello board for check in purposes.

## 2.4 Product Backlog

The following is a list of the product backlog for this project as a whole. The sprint backlogs are laid out in more detail in the project chapter or the sprint reports of this document. The backlogs for this project were tracked using Trello a web interface for project management. During the development process both the team and the client will have access to the board to view the progress of the current iteration of the project. During development the project will consist of six sprints of roughly three weeks each with a week after each for review as mentioned above. The project's source control is contained within a Github repository provided by the South Dakota School of Mines and Technology.

### 2.4.1 Sprint 1 Backlog

- Set Up Github repository
- Conduct Programming Language Research and Analysis
- Conduct Database System and Infrastructure Research and Analysis
- Conduct GUI Interface and Framework Research
- Sprint 1 Research and Sprint Report and Decision
- Begin Practicing and Learning Development Materials
- Prepare Client Presentation 1



### 2.4.2 Sprint 2 Backlog

- Create Starting Database Tables and Infrastructure
- Create GUI Interface Theme and Design
- Create Log In Page
- Create Permission System
- Create Landing Pages for Admin and Teachers
- Sprint 2 Report and Analysis

### 2.4.3 Sprint 3 Backlog

- Create Student Search
- Add a Class
- Produce a Class Role Sheet
- Create Employee Search
- Create Class Search
- Add Advanced Search to Searches
- Add and Remove Students From a Class
- Modify Student Information
- Assign Teacher to a Class
- Sprint 3 Report and Analysis
- Turn In Semester One Documentation
- Prepare Client Presentation 2

### 2.4.4 Sprint 3.5 Backlog

Most of Sprint 3.5 was bug fixes and putting functions together in the interface. After this sprint it became clear to the team that the team would not be able to complete the student interface during this first iteration of the project. As such the student interface and the considering functionally were removed from the list of requirements upon discussions with the client.

- Role Sheet Redesign
- Tie the individual Functions Together Into Single Interface and Prototype
- Fix Various Bugs
- Add In Extra Functions Implied By User Stories
- Adapt Database After First Semester

### 2.4.5 Sprint 4 Backlog

- Enter Staff Pay Rates
- Enter and Update Tuition Rates
- Apply and Update Credits to a Student
- Give Early Registration Discounts
- Billing/Payment History for a Student
- Enter a Full Payment for Several Students
- Full Payment for One Student
- Allow for Payments From Multiple Sources
- Look at What A Student Still Owes
- View Teaching History
- Sprint 4 Report and Analysis

### 2.4.6 Sprint 5 Backlog

- Enter in Student Registration Information for Existing Students
- Modify Admin and Teacher Crossover
- Ignore Address Case to Forms
- Modify Add/Remove Students for Client Update
- Allow for and Add Multiple Source of Pay to Adapt to Client Request
- System Admin List
- Enter Staff Hours
- Add and Remove Class Location
- Remove Admin
- Remove Class
- Remove Teacher
- Remove Student
- Prorated Refunds
- Update Tuition Rates Request By Client
- Order List Functions
- Password and Username Reset Functions
- Quality Updates
- Client and Teacher Project Reformation Meeting
- Sprint 5 Report and Analysis

### 2.4.7 Sprint 5 Fixed Bug Backlog

- Add Remove From Teacher-Class Table to Remove Teacher Function
- Fixed Searches Errors
- Fixed Advanced Search Issues
- Remove Open Buttons
- Add More Return Buttons for Quality
- Added Dynamic Teacher and Student Schedule
- Modified Format of Functions to Better Match Database Options
- Remove Extra Navigation Bar
- Remove and Update Extra System Buttons
- Error Check Bug Fixes
- Fixed Print Functionality After Change
- Quality Updates

### 2.4.8 Sprint 6 Backlog

- Change Log In and Highlighted Button
- Compute Instructor Wages
- Add Student and New Student Registration
- Update to Refund Page
- Assign Discounts (if possible before submission)
- Modify Location
- Remove Command Prompt
- Final Mass Test for This Iteration of Project
- Design Fair Materials and Presentation
- Reformatting and re Factoring of Design Docs for Iteration Model
- Finish as much material for next Iteration as Possible
- Client and Teacher Project Evaluation Meeting
- Sprint 6 Report and Analysis
- Submission of Iteration Materials

## 2.5 Research or Proof of Concept Results

Before production could begin research had to be conducted into which programming language, GUI framework, and database type would be used to complete the project. A explanation of the research conducted can be found in the sprint one wrapper or in the prototype sections of this document. After this research was conducted the team selected Python, PyQt, and MySQL as the language, GUI, and database respectfully. After the research no explicit proof of concept was required, however as pages are functional pages are shown to, or approved by the client. With the main goal of this team being to develop as much of the project as possible to hand over to the client for either the next iteration or other course of action, decided by the client.



## 3

---

### Project Overview

---

This section provides some information with regards to the team roles, project management, and phase overview.[MB]

#### 3.1 Team Members and Roles

The DanceSoft team consist of two members, Marcus Berger and Dicheng Wu.

Marcus Berger(Scrum Master/Development Team) - As a member of a two person team the roles for this project blend together significantly. Both team members mostly do equal shares of all work types. As the primary manager of the DanceSoft Trello board, Marcus had mostly taken on the role of scrum master within the group. However his primary role is still development of DanceSoft.

Dicheng Wu(Development Team) - Dicheng's primary role as with both members of the team, is development of the software. However like the other members since the team is so small each member of the two person team must be able to fill all needed roles within the team.

Dr. Jeff McGough(Product Owner) - While not a working member of the team Dr. McGough is a secondary scrum master and product owner to the group due the small size of the team. Main duties in this role include talking to the client about what is needed, and making sure the team stays on task and is going in the correct direction based on the clients needs .

Author notation will be identified by [MB] for Marcus and [DW] for Dicheng at the start of major sections (ex: after heading 2.0 above). Subsections within the main sections assumed to be by the same author. [MB]

#### 3.2 Project Management Approach

The client requires a weekly meeting every Wednesday at 2:00 p.m. to check on the progress of the system during the fall 2015 semester at South Dakota School of Mines and Technology. During the spring 2016 semester the team meeting changed to Tuesdays at 2:00 p.m. and Thursdays at 3:00 p.m. if necessary. These meeting vary on topic and length depending on the needs of the project and the status of task. The senior design class requires that this project be completed in six sprint that are all roughly three weeks long, with a week long results period after each one. Another project requirement is the presentations that are required by the senior design class. These presentation occurs twice every semester usually after the first and last sprint each semester. Each presentation cover the content of the project up to that point, and updates on topics such as risk mitigation, budget, and current prototypes. Lastly it was requested by Dr. McGough as part of senior design and as the client that we provide him with access to the Github repository for the project and the Trello board for check in purposes.

The internal team management was mostly was mostly done by Marcus Berger and Dr. McGough. Marcus used the free service Trello to manage the tasks that the team needed to complete. As the project progressed the team had to reevaluate the management approach as the DanceSofts project requirements changed. After sprint 3 it was decided that due to time left for the project that the management of the student interface would need to be finish in a future iteration.

### 3.3 Stakeholder Information

The stakeholder and sponsoring customer for this project is Dr. Jeff McGough a computer science professor at the South Dakota School of Mines and Technology and the vice president of the Academy of Dance Arts in Rapid City, South Dakota.

Well not the sponsor of the project Dr. McGough's wife, Julie McFarland is also a key part of the customer base and a stakeholder as the owner and artistic director of the Academy. Other academy members while not directly related also share a stake in the project development as the software directly processes and modifies data given to the academy. Students and teachers will not have the access to direct evaluation of the various iterations of the DanceSoft project. However as the project nears completion in future senior design classes or however the clients choose to proceed, these groups could be affected by the uses of and updates to the software. Therefore they have a secondary stake in the project's various iterations.

#### 3.3.1 Customer or End User (Product Owner)

The primary end user for this product is Julie McFarland and her employees to manage the Academy of Dance Arts. Julie will not be playing a direct role in product development but is able to convey the academy's needs through Dr. McGough. Dr. Jeff McGough is the primary point of contact in the project. He assumes the role of scrum master at times and drives the product backlog while providing more details on product backlog materials during the weekly meeting with the development team.

#### 3.3.2 Management or Instructor (Scrum Master)

Dr. Jeff McGough is the primary point of contact in the project. He assumes the role of scrum master at times and drives the product backlog and provides more details on product backlog materials during the weekly meeting with the development team. He also acts as the assessment for project progress. This means that as the project progressed Dr. McGough is able to reevaluate and reestablish requirements as the project needed or was requested by the client.

#### 3.3.3 Developers –Testers

The DanceSoft team consists of two members, Marcus Berger and Dicheng Wu, who are both primarily developers and testers. Due to the fact that the team is only two members, all development roles are shared between the two team members. The team's job is to develop, test, fix and adapt the various requests and requirements given by the client/scrum master Dr. McGough. As developers the team takes the user stories and develops a backlog for each sprint. Then the team's job is to take these user stories and produce the code needed to complete each backlog entry.

After the backlog entry has been completed it is the developers' job to test and error check the functions to make sure that the function completes all its needed tasks, in the way the team wants. Also testing is done to make sure the files don't create any problems with the rest of the software. Testing is covered in more detail in the testing section of this document.

### 3.4 Budget

There was no budget or monetary compensation for this project. When the project was initially laid out there was an idea for a budget that would be used for a Linux box. The Linux box would have been used to store the system on a local device. However as the requirements were brought down and redefined to an iteration of the project, rather than a full complete project, it became clear that the project would not reach a deployment state where a local machine was necessary. However the client could provide a Linux box for project storage at the end of the semester, which will not be claimed as a budget by this project. As work continues on this project the budget may grow. Most expenditures associated with the Senior Design class, an example being the DanceSoft senior design poster, were covered by the DanceSoft team members. Monetary compensation for this project follows the guidelines laid out in the DanceSoft Software Contract. The team will be given and accept no form of compensation for the work on the DanceSoft or any related project. This is in accordance with the conditions of the project, client, and the senior design class.

## 3.5 Intellectual Property and Licensing

The intellectual property for this project is currently the property of the client Jeff McGough. However due to Dr. McGough's involvement with the South Dakota of Mines and Technology, and the senior design class, the South Dakota Board of Regents policies must be considered. These policies require that any work done by a teacher or for a teacher by a group of one or more students be submitted as property of the South Dakota Board of Regents.

The only way to avoid this policy and prevent the Academy software from being owned by the state is to make the project open source in accordance with the board's policy. As a result of this set of policies all code worked on by the DanceSoft team is stored in a public Github repository provided by SDSMT for the senior design class.

Another intellectual property and licensing policy that must be considered as part of the teams choice to use PyQt is the GPL (general public license), under which the PyQt software falls. This license requires that the user release all source code for a project using any GPL licensed code in their software. However this is only required if the client plans on distributing the software they have created. As such this license will not cause any issues for Dr. McGough or the Academy of Dance Arts since the client does not plan on distributing the code to anyone other than themselves.

Based on the goals of the project the team has constructed as software that does not violate any intellectual property rules or regulations. Therefore the current rights to the software belong solely to the client until such a time as the client chooses to distribute the project for monetary gain.

## 3.6 Sprint Overview

This project will be implemented in phases, the phases follow. [MB]

### 3.6.1 Sprint 1: Initial User Story and Requirements Gathering

This phase consisted of meetings and discussing the user stories and requirements with the product owner. The product owner also laid out limitations and constraints for the project. More information can be found in section **3.0 Requirements**

Also tackled during this phase was the research conducted into the tools the team were going to use for the project. There were three main areas the team needed to tackle to decide the frameworks for the project. First the team needed to pick the programming language that would be used for the project. Two main languages were analyzed, Python and Swift. Swift is the Mac native language recently released by Apple Inc. the benefits found for Swift were its nativity to the Mac OSX and it would be the easiest to integrate into Mac. However since Swift was relatively new the team would have to take even more time to learn the new language, and since the client did not know Swift it would be harder to analyze and make adaptations to in the future. Python had the advantage of being a language that is more universally known, both to the team, the client, and the online community that the team could turn to for support or questions. Due to this and a few other factors laid out in more detail in **Sprint Report 1** the team choose Python as the programming language for the projects primary programming language.

The other two areas of research were database and interface framework. The team looked to PyQt, Kivi, Tkinter, and other graphical interface frameworks. The team decided after looking at each one that due to the ease of use and some companies listing GUI experience, that the team would use PyQt as the framework for the interface. Then the DanceSoft team looked into database software. After looking at both SQL and non-SQL database options, such as MySQL and Mongo the team went with a MySQL database for the storage. As with programming language, the research results are laid out in **Sprint Report 1**

### 3.6.2 Sprint 2: Database Creation and Starting Pages

During this phase the database schema was constructed and implemented, being sure to keep the schema as concise as possible. The goal was to generate a database that could effectively support the needed functionality and GUI connectivity. After the database was constructed, the team moved on to the starting landing pages which are jumping off pages for functionality creation. These page are modified as phases progress and further functionality was added.

During this phase the team also plotted out the first drafts of the overall user interface flow for the desktop GUI. The team later modified this as the projects requirements were flushed out. The team would also reexamine this structure in later sprints as certain requirements were dropped, added or modified. Once the first version of the database was up and running within the School of Mines' MySQL sever the team began working on the log in page, and the main navigation pages for both Academy admins and teachers.

The log in page is used to confirm the user of the system and check whether or not the user has admin permissions. This system is required because of the different features the admins can access within the system, such as changing pay rates or adding students or teachers to the system. The log in page then directs to the landing page selection where the user, if they have permission, can access the admin or teacher interfaces.

The first drafts of the admin and teacher interfaces were implemented in this phase as well. The admin interface contained buttons through which the user can access the various functionality of the interface, These include "Manage Employees" which takes the user to most of the employee related features within the project, also the "Manage Classes", "Manage Students", and "Manage Billing and Payroll" buttons take the user to the class, student, and billing and payroll features respectfully. The teacher interface like the admin interface contains buttons for managing students and classes, however these features differ slightly between interfaces since admins have more system access. The teacher interface also contains a "Personal Information" button which allows the user to modify certain aspects of their information in the system, such as their hours, or their user-name and password.

### 3.6.3 Sprint 3: Functionality Creation

This phase will compose the bulk of the project as the first development of the functionality requested by the project owner are constructed. As the prototype progresses check ins with the client will be conducted to be as sure as possible that the team stays on track. Pages will be tested as the pages are constructed.

During this phase several pages were constructed. The first of these pages was the "Add a Class" page, this pages creates a form that the user can type the information for the class in and create an entry in the database for that class.

The second page made was the add student page which allowed the user set a students registration status for specific classes to approved, or rejected from pending. This function was later pulled when the student interface was dropped from the requirement. However at the request of the client the function was left in for future adaptations and uses.

Another set of pages during this phase was the search pages. There are three types of search pages within the project, student, class, and teacher. The user is able to search any of these elements by name with the default search bar. Users can also search by different fields if they use the advanced search functionality. Lastly the users can select specific data fields and pull up all that entry's data and update it if needed.

Assign teacher allows users to select a class, if that class is assigned to a teacher already a dialog pops up asking the user if they want to reassign the class. If the class is not assigned or if the user selected to reassign the class then the function generates a list of available teacher for the given class time. The user can then select a teacher and assign them.

The log in page allows the system to deal with different user levels, and provide navigation to the different landing pages. The role sheet allows the currently signed in teacher to produce and print the role sheets for each of their classes. The last main function created during this phase was the update teacher function which allow the admins to select a teacher then a form is population with the selected teacher's information. The user can then update the teacher's information within the system. The final step of this phase was the third client presentation, which capped of the first semester and the senior design I class.

### 3.6.4 Sprint 4: Development Updates and Payroll/Billing

This phase consisted of updates to phase three functions, adding new functions, and adding in the payroll function to the project. The first part of this phase was sprint 3.5, during this time the team added several bug fixes such as assign class and connecting functions together for a connected prototype.

One of the pages completed during this sprint was the teaching history function. This function displays all the classes the teacher has taught in the recent time frame. Another page tackled was the student billing invoice in this function the amount owed and the amount the student has payed is displayed along with



Name	<input type="text" value="Tap II"/>
Cost	<input type="text" value="30.00"/>
Start Time	<input type="text" value="16:45:00"/>
End Time	<input type="text" value="17:30:00"/>
Day	<input type="text" value="Tuesday"/>
Location	<input type="text" value="Rapid City"/>
Cap	<input type="text" value="30"/>
Clothing	<div>Dress Code for Girls: black jazz pants, tap oxfords, camisole, or leotard top Boys: black or white tee, black jazz pants &amp; black tap oxfords.</div>
Description	<div>Music &amp; drumming first, movement follows. Rhythm, balance &amp; coordination skills are vital to success, as we explore history of this percussive movement form, born in America with roots firmly planted in Africa! We cover early time steps and fundamental-through-advanced repertoire and improvisation.</div>
Start Date	<input type="text" value="1/18/2016"/>
End Date	<input type="text" value="5/7/2016"/>
Age	<input type="text" value="6"/>
Age Limit	<input type="text" value="0"/>

\*Required Fields      \* 0 in Age or Age Limit represents no restriction (Leave 0's for All age classes)

Figure 3.1: Add Class Form

the classes the student took to generate that amount. Alongside this function the team created the ability to process a payment for a student. The user is allowed to enter in the amount the student paid and the payment type. The user can also process a full payment if the student elected to pay the full amount.

As a payment is processed it is added to the student billing history which can then be viewed and printed by the user. The billing history contains the payment id, the name of the student, the amount paid, and the date of payment. Another function implemented during this phase was the students credit interface, within this page the user can select a student and then apply a credit to that student. The credits do not directly connect to any other function due to the fact that the Academy handles credits on a student by student basis at the discretion of the Academy owner Julie McFarland.

Similar to credits the system also includes the functionality for the modify tuition and fees within the database. The tuition page allows the user to select the tuition name and change the rate. The user can also hit the add button on the page which opens a dialog box where they can put in the required information. Other buttons included within this page are update and delete which allow the user to update an existing entry or delete an entry respectfully. The fees page has the same structure as the tuition pages, the user is able to select a fee and update or delete the fee rates. There also is the ability when add or updating a fee to mark it as a percent so the value given must be a decimal from zero to one. The last function added during this phase was the ability to enter in teacher pay rates, this function allows the admin in the system to declare a pay name and pay rate that a teacher has and the amount of hours they worked. The function then calculates the gross wage of the teacher based on those pay rates. The teacher side also has an enter hours pages which allows the user to enter in the amount of hours they worked under each pay type.

### 3.6.5 Sprint 5: Updates Bug Fixes and Functionality

During sprint five the team tackled the remain functionality, the sprint rollover and various bug fixes. The first of these fixes was the ability to update an admin or teachers information and have the updates crossover between the two tables in the database. The address in the forms page where also modified to always be uppercase to avoid the occurrence of two similar names within the database. Another modification made was to the approve/reject student page, when the page was initially created the team did not allow rejected students to be re approved, this function was then adapted to allow for this to be closer to the client's request. Change user credentials were also added during this phase.

An admin list was added that displays the admins for the system and allows the addition and removal of admins from the system. Alongside this function several removal functions where added. These include: remove student, remove class, remove class location, and remove teacher, the function are final purges for the data. This means that if a user removes a teacher the teacher information, classes taught, classes assigned, and account information are all removed from the system. For students the delete also includes all the billing and student data, so the system delete functions should only be used when the user is absolutely sure that the information is not longer needed.

The main function completed during this phase was the student registration interface. This allows the Academy to enter in new students to the system, and update the student. The user can then pull up a list of available classes and classes the student is regesteer for and add them to classes. These classes are then added to the student-class table and the classes are processed by the system. The user can then view and print out student and or teacher schedules.

Bug Fixes Completed During Phases 4, 5, and 6:

1. Date time update bug
2. Form information bug
3. Add teacher class to remove teacher
4. Spelling errors
5. Remove class cost
6. Search update bug
7. Search refresh bug

8. Advanced search bug
9. Error checks
10. Remove uneccicayry button
11. Dynamic times on schedules
12. Few combo box changes
13. Text changes

### 3.6.6 Sprint 6: Updates, Fixes, and Iteration Packaging

During the last phase the team attempted to finish as much as possible before turning in this iteration of the project. This phase consisted mostly of bug fixes and client updates after demoing the final project the team managed to create. First since the remove student showed all students in the system the team added a search bar to clean up the functions execution as much as possible. Second entering teacher hours was refined a bit to allow for a default course hours rate. Next several bugs were fixed, and modify location functionality was added.

Student registration was also complete during this phase. The last half of this phase was focus on prep and execution of the design fair presentation held by the South Dakota School of Mines and Technology. Overall this last phase consisted of the last step of senior design and preparing to hand off our teams iteration of the DanceSoft project.

## 3.7 Terminology and Acronyms

1. Backlog - A list of task to be completed
2. Budget - money provided by the client to supply the needed materials for the project
3. Database - A storage and platform to manipulate data for the projects
4. GUI - Graphical User Interface - the front end screen that the users interact with using graphical assets
5. Sprint - three week time periods where portions of the project are completed
6. Timeline - the plan of when a assignment is to be completed
7. GPL - General Public License - a type of software development license
8. SDSMT - the South Dakota School of Mines and Technology's university acronym
9. SDBOR - South Dakota Board of Regents
10. Interface - The screen or set of screens that the user can see and navigate within the project
11. Source Control - A system used to manage multiple people working on the same collections of information to maintain consistency,

## 3.8 Sprint Schedule

There are three sprints for Fall semester and three for Spring semester

1. Sprint 1: 9/14/15 - 10/2/15
2. Review and Client Presentation: 10/20/15
3. Sprint 2: 10/12/15 - 10/30/15
4. Sprint 3: 11/9/15 - 11/27/15

5. Review and Client Presentation: 12/3/15
6. Sprint 4: 1/18/15 - 2/5/16
7. Sprint 5: 2/15/16 - 3/4/16
8. Review and Client Presentation: 3/22/16
9. Sprint 6: 3/21/16 - 4/15/16
10. Design Fair: 4/19/16

## 3.9 Backlogs

### 3.9.1 Sprint 1 Backlog

- Set Up Github repository
- Conduct Programming Language Research and Analysis
- Conduct Database System and Infrastructure Research and Analysis
- Conduct GUI Interface and Framework Research
- Sprint 1 Research and Sprint Report and Decision
- Begin Practicing and Learning Development Materials
- Prepare Client Presentation 1

### 3.9.2 Sprint 2 Backlog

- Create Starting Database Tables and Infrastructure
- Create GUI Interface Theme and Design
- Create Log In Page
- Create Permission System
- Create Landing Pages for Admin and Teachers
- Sprint 2 Report and Analysis

### 3.9.3 Sprint 3 Backlog

- Create Student Search
- Add a Class
- Produce a Class Role Sheet
- Create Employee Search
- Create Class Search
- Add Advanced Search to Searches
- Add and Remove Students From a Class
- Modify Student Information
- Assign Teacher to a Class
- Sprint 3 Report and Analysis
- Turn In Semester One Documentation
- Prepare Client Presentation 2

### 3.9.4 Sprint 3.5 Backlog

Most of Sprint 3.5 was bug fixes and putting function together in the interface. After this sprint it became clear to the team that the team would not be able to complete the student interface during this first iteration of the project. As such the student interface and the considering functionally were removed from the list of requirements upon discussions with the client.

- Role Sheet Redesign
- Tie the individual Functions Together Into Single Interface and Prototype
- Fix Various Bugs
- Add In Extra Functions Implied By User Stories
- Adapt Database After First Semester

### 3.9.5 Sprint 4 Backlog

- Enter Staff Pay Rates
- Enter and Update Tuition Rates
- Apply and Update Credits to a Student
- Give Early Registration Discounts
- Billing/Payment History for a Student
- Enter a Full Payment for Several Students
- Full Payment for One Student
- Allow for Payments From Multiple Sources
- Look at What A Student Still Owes
- View Teaching History
- Sprint 4 Report and Analysis

### 3.9.6 Sprint 5 Backlog

- Enter in Student Registration Information for Existing Students
- Modify Admin and Teacher Crossover
- Ignore Address Case to Forms
- Modify Add/Remove Students for Client Update
- Allow for and Add Multiple Source of Pay to Adapt to Client Request
- System Admin List
- Enter Staff Hours
- Add and Remove Class Location
- Remove Admin
- Remove Class
- Remove Teacher

- Remove Student
- Prorated Refunds
- Update Tuition Rates Request By Client
- Order List Functions
- Password and User Name Reset Functions
- Quality Updates
- Client and Teacher Project Reformation Meeting
- Sprint 5 Report and Analysis

### 3.9.7 Sprint 5 Fixed Bug Backlog

- Add Remove From Teacher-Class Table to Remove Teacher Function
- Fixed Searches Errors
- Fixed Advanced Search Issues
- Remove Open Buttons
- Add More Return Buttons for Quality
- Added Dynamic Teacher and Student Schedule
- Modified Format of Functions to Better Match Database Options
- Remove Extra Navigation Bar
- Remove and Update Extra System Buttons
- Error Check Bug Fixes
- Fixed Print Functionality After Change
- Quality Updates

### 3.9.8 Sprint 6 Backlog

- Change Log In and Highlighted Button
- Compute Instructor Wages
- Add Student and New Student Registration
- Update to Refund Page
- Assign Discounts (if possible before submission)
- Modify Location
- Remove Command Prompt
- Final Mass Test for This Iteration of Project
- Design Fair Materials and Presentation
- Reformatting and re Factoring of Design Docs for Iteration Model
- Finish as much material for next Iteration as Possible
- Client and Teacher Project Evaluation Meeting
- Sprint 6 Report and Analysis
- Submission of Iteration Materials

## 3.10 Development Environment

The basic purpose for this section is to give a developer all of the necessary information to setup their development environment to run, test, and/or continue development of the DanceSoft project, while also providing information on the systems the team used to create the project.

### 3.10.1 Source Code Development

The source code for this project was written in Python. Due to the cross platform nature of this language the team used a variety of different IDEs over the course of development, these included:

1. Python IDLE
2. Microsoft Visual Studio 2015
3. Xcode

Any developers wishing to continue work on this project should be able to run this software within any Python 3 capable interface of their choice.

### 3.10.2 MySQL Database

The projects back end contains a MySQL database. MySQL can be installed on a developers computer free of charge off of the MySQL website. This package also contains a helpful GUI interface called MySQL workbench which can aid the user in manipulating the database if they so choose. However the system should be editable within any MySQL enabled database manipulation software. Once a developer has MySQL installed they simply need to log in and connect to the database using credentials provided by the client.

### 3.10.3 PyQt

The projects front end interface is developed using PyQt. PyQt is an extension of Qt it allows the development of Qt GUI's and functionality in Python. The tool kit contains all the normal Qt widgets including: line edits, spin boxes, combo boxes, text edits, and other normal GUI widgets. PyQt also has a designer that the team used to create the major interface pages. The designer allows the user to click and drag components and develop the pages in a more visual environment. The designer can then create generated code for the pages using the `pyuic4` command in a terminal as follows, "`pyuic4 -o generatedCode.py uiFilename.ui`" this command produces a .py file containing the generated designer code. The generated class can then be included in the python files to use the code. In classes written by a developer, programs can include any of the major Python 3 or PyQt libraries.

Some common libraries the team used where:

- QtGui
- QtCore
- PyQt4.QtSql

## 3.11 Development IDE and Tools

The two main IDEs were used in the this project were Microsoft Visual Studio 2015, and Xcode 6. Python IDLE would also be used on occasion for minor quick fixes so the main IDEs would not need to be loaded completely. Of these the bulk of development was conducted with Visual Studio due to the fact that the laptops provided by South Dakota School of Mine and Technology run windows as their primary operating system. Visual Studio also provided a suite of debugging and testing features that allowed the team to manage and manipulate the code effectively.

The second IDE used was Xcode which is the primary development environment for the Mac operating

system. This IDE was used when ever we want to directly test Mac compatibility with our code. Since Mac is the required working operating system for this project. Though due to accessibility Xcode was not the Main IDE used by the team. Should the project be further developed in the future IDE selection should not matter due to the cross-platform development of the project.

1. Visual Studio install: <https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx>
2. Visual Studio Reference: <https://msdn.microsoft.com/en-us/library/scesz732.aspx>
3. Xcode install: <https://developer.apple.com/xcode/downloads/>
4. Xcode Reference: <https://developer.apple.com/>
5. IDLE install: comes with python 3 download url<https://www.python.org/downloads/>
6. IDLE Reference: <https://docs.python.org/3.1/>

### 3.12 Source Control

Source control for this project was conducted using Github, and the Github GUI. Github is a git add in, a developer could use whatever git manager they want. The Github GUI can be installed from [www.github.com](http://www.github.com). The repository was provided by South Dakota School of Mines as part of the senior design class. A developer wishing to continue work on this project simply need a git software and access as a contributor to the repository. However once the clients move the code to a local device a developer will need access to this device as well.

### 3.13 Dependencies

Currently the project contains two known dependencies. The first is in the PyQt4 Python library which if changed could cause issues within the system GUI. However this seem unlikely since the main focus of PyQt updates is now focused on PyQt5. Second is the project dependence on MySQL relational database. This dependency should also be negligible since any update to MySQL are normally done with continuing compatibility with existing software in mind.

### 3.14 Build Environment

A user, or developer can build the project through the use of compiled python scripts. A user can run the log in script which will compile some of the Python scripts as to speed up various aspects of running the project. There are no special build scripts that are requires before the project can be run a user simply needs to run the login script for the DanceSoft project.

### 3.15 Development Machine Setup

1. A machine will need to be acquired that is capable of running PyQt and MySQL
2. The user will need to install python 3 from <https://www.python.org/download/releases/3.4.3/>
3. Next the user need to install the PyQt 4 library and the Qt designer if necessary from <https://riverbankcomputing.com/software/pyqt/download>
4. After downloading Pyqt the user need to download MySql from <http://dev.mysql.com/> and connect to the database where the DanceSoft database is stored.
5. Now if the PyQt libraries have been installed correctly the user should be able to run the PyQt code and proceed to develop the DanceSoft project.



## 4

---

# Design and Implementation

---

This section is used to describe the design details for each of the major components in the system. Note that this chapter is critical for all tracks. Research tracks would do experimental design here where other tracks would include the engineering design aspects. This section is not brief and requires the necessary detail that can be used by the reader to truly understand the architecture and implementation details without having to dig into the code. Sample algorithm: Algorithm 1. This algorithm environment is automatically placed - meaning it floats. You don't have to worry about placement or numbering.

---

**Algorithm 1** Calculate  $y = x^n$

---

**Require:**  $n \geq 0 \vee x \neq 0$

**Ensure:**  $y = x^n$

```
 $y \leftarrow 1$ 
if  $n < 0$  then
   $X \leftarrow 1/x$ 
   $N \leftarrow -n$ 
else
   $X \leftarrow x$ 
   $N \leftarrow n$ 
end if
while  $N \neq 0$  do
  if  $N$  is even then
     $X \leftarrow X \times X$ 
     $N \leftarrow N/2$ 
  else  $\{N \text{ is odd}\}$ 
     $y \leftarrow y \times X$ 
     $N \leftarrow N - 1$ 
  end if
end while
```

---

Citations look like [2, 1, 3] and [6, 4, 5]. These are done automatically. Just fill in the database `designrefs.bib` using the same field structure as the other entries. Then `pdflatex` the document, `bibtex` the document and `pdflatex` twice again. The first `pdflatex` creates requests for bibliography entries. The `bibtex` extracts and formats the requested entries. The next `pdflatex` puts them in order and assigns labels. The final `pdflatex` replaces references in the text with the assigned labels. The bibliography is automatically constructed.

## 4.1 Architecture and System Design

This is where you will place the overall system design or the architecture. This section should be image rich. There is the old phrase *a picture is worth a thousand words*, in this class it could be worth a hundred points

(well if you sum up over the entire team). One needs to enter the design and why a particular design has been done.

#### 4.1.1 Design Selection

Failed designs, design ideas, rejected designs here.

#### 4.1.2 Data Structures and Algorithms

Describe the special data structures and any special algorithms.

#### 4.1.3 Data Flow

#### 4.1.4 Communications

#### 4.1.5 Classes

#### 4.1.6 UML

#### 4.1.7 GUI

#### 4.1.8 MVVM, etc

### 4.2 Major Component #1

#### 4.2.1 Technologies Used

This section provides a list of technologies used for this component. The details for the technologies have already been provided in the Overview section.

#### 4.2.2 Component Overview

This section can take the form of a list of features.

#### 4.2.3 Phase Overview

This is an extension of the Phase Overview above, but specific to this component. It is meant to be basically a brief list with space for marking the phase status.

#### 4.2.4 Architecture Diagram

It is important to build and maintain an architecture diagram. However, it may be that a component is best described visually with a data flow diagram.

#### 4.2.5 Data Flow Diagram

It is important to build and maintain a data flow diagram. However, it may be that a component is best described visually with an architecture diagram.

#### 4.2.6 Design Details

This is where the details are presented and may contain subsections. Here is an example code listing:

```
#include <stdio.h>
#define N 10
/* Block
 * comment */
```

```
int main()
{
    int i;

    // Line comment.
    puts("Hello world!");

    for (i = 0; i < N; i++)
    {
        puts("LaTeX is also great for programmers!");
    }

    return 0;
}
```

This code listing is not floating or automatically numbered. If you want auto-numbering, but it in the algorithm environment (not algorithmic however) shown above.

## 4.3 Major Component #2

### 4.3.1 Technologies Used

This section provides a list of technologies used for this component. The details for the technologies have already been provided in the Overview section.

### 4.3.2 Component Overview

This section can take the form of a list of features.

### 4.3.3 Phase Overview

This is an extension of the Phase Overview above, but specific to this component. It is meant to be basically a brief list with space for marking the phase status.

### 4.3.4 Architecture Diagram

It is important to build and maintain an architecture diagram. However, it may be that a component is best described visually with a data flow diagram.

### 4.3.5 Data Flow Diagram

It is important to build and maintain a data flow diagram. However, it may be that a component is best described visually with an architecture diagram.

### 4.3.6 Design Details

This is where the details are presented and may contain subsections.

## 4.4 Major Component #3

### 4.4.1 Technologies Used

This section provides a list of technologies used for this component. The details for the technologies have already been provided in the Overview section.

#### **4.4.2 Component Overview**

This section can take the form of a list of features.

#### **4.4.3 Phase Overview**

This is an extension of the Phase Overview above, but specific to this component. It is meant to be basically a brief list with space for marking the phase status.

#### **4.4.4 Architecture Diagram**

It is important to build and maintain an architecture diagram. However, it may be that a component is best described visually with a data flow diagram.

#### **4.4.5 Data Flow Diagram**

It is important to build and maintain a data flow diagram. However, it may be that a component is best described visually with an architecture diagram.

#### **4.4.6 Design Details**

This is where the details are presented and may contain subsections.

## 5

---

# System and Unit Testing

---

This section describes the approach taken with regard to testing the DanceSoft project.

### 5.1 Overview

The general testing for this iteration of the DanceSoft was done through manual user testing of the GUI and making sure that that correct values ended up in the database in the way the team was expecting. The team also checked removal to make sure that they were being processed correctly. Other GUI functionality was also tested such as buttons and overall usability of functions.

Success for this testing procedure is measure by correct interactions with the database and user. An example of a successful test would be a user adds a teacher, the system check required fields and outputs an error dialog if not filled out. Once the form is filled out the teacher information is correctly placed in the Teacher table and an account is made within the Account table.

Failure is similarly defined for this project, if a function misplaces or incorrectly places data in the database or if the query fails and breaks search, then it is defined as a failure. Other instances of failure are defined for the individual functions.

### 5.2 Dependencies

There are several dependencies for the testing of the DanceSoft system. The follow is a general overview of the dependencies found by the team:

- Team Inexperience - The team lack mass testing experience due to the fact that neither team member has had any industrial experience before this project.
- Difficulty of Unit Testing - Owing to the project being mostly GUI and Database the team had issues with developing unit test for the project. So for many reason the project was manual and within database testing
- Team Testing Effectiveness - Since the testing was manual the thoroughness of testing was completely determined by the team members efficiency. This caused issues within the team and the testing since some team member were less effective at times.

### 5.3 Test Setup and Execution

The DanceSoft project has all of it functions tied to various GUI elements such as buttons, spinboxes, and line edits. As such the teams testing approach was to test the functions and database connections. Most pages follow this general testing structure:

- Test error notifications - such as not filling out required form fields

- Button allocence - can the user only click a button or element within the page when allowed.
- Correct Data Apperence - Does the functions that recive data recive the correct values from the database
- Correct Data Submiition - Does data creation or updates get processed within the database and placed in the correct tables
- Correct Database Handling - Are the updates place in the correct places and are they complete

## 5.4 System Testing

Login page:

Log in testing case structure:

- test invalid user name
- test invalid password
- test correct log in
- test log in button directing to landing selection page
- test cancel button exiting software
- test permission system - once at landing selection is someone with the correct permissions locked out of admin landing
- test navigation - do the buttons to the admin and teacher landing pages work navigate correctly

Landing Selection:

- Tested to confirm that the admin button goes to the admin landing page
- Tested to confirm that the teacher button goes to the teacher landing page

Admin Landing:

The admin landing page is composed of entirely buttons that go to the function windows tied to the button. Therefore the admin landing testing is covered in system integration testing and testing other function since the buttons needed to work to test the functions.

Teacher Landing:

As with the admin landing page the testing for this page is retroactively covered by navigating to other pages through the GUI, the system integration testing and the demos.

Add Information Forms:

There are several forms that add information to the system these include add teacher, add student, and add class the testing structure follows:

- Test that each button connected to a form displays correctly and opens the correct form
- Test the line edits ability to accept input
- Test the spin boxes
- Test the date edits
- Test Combo boxes

- Test the submit button triggering the require field dialog if not filled out correctly
- Test the submission of new data entry to the database when the submit button is clicked

The teacher and student forms error check the name, phone numbers which must be formatted as a ten digit number, and address. The address is placed in the data base in caps to prevent the duplication of addresses within the database. As a class is generated the class id is set to the next in the database.

The student forms include a guardian option where the user is able to select a guardian from a combo box or add a new one to the system. Tests were run to make sure that

Search Pages:

Show Admin List:

Update Teacher Information:

Enter Teacher Hours:

Assign Teacher to Class:

Teaching History:

Set Semester:

Student Credits:

Add/Remove Student:

Registration:

Removal Functions:

Enter Partial Payment:

Enter Full Payment:

Enter Teacher Pay Rate:

Student Balance:

Billing History:

Tuition Rates and Fees:

Student Schedule:

Teacher Class Schedule:

Class Role Sheet:

Change Password

Change Username:

Modify Personal Information:

Enter Hours:

## 5.5 System Integration Analysis

System integration testing occurred at several points during development with the goal of making sure that the separate functions of the user interface tied together and worked correctly.

At the start of sprint 2 the admin and teacher main pages were tied into the "login" page, through several log ins the team was able to confirm that the navigation works as intended.

The major system integration test occurred during sprint 3.5. During this time the majority of the functions were tied to the main pages. These pages include the searches, assign teacher, and many others completed before this point. Once the pages were connected the system was ran several times to confirm that the navigation connected to the correct pages and the functions ran. This phase created the projects first connected prototype, which culminated in a first semester prototype demo for the client after returning from sprint 3.5.

After the first connected prototype was created the team tied the functions in as they were finished and through testing the functions also tested system integration since the team would need to log in and navigate to the various pages within the system.

Once development of the project was frozen by the client the team conducted a last round of system integration testing was conducted across all functions to confirm that the system would correctly flow for the South Dakota School of Mines Design Fair. After the design fair the team has conducted no further test due to the development freeze on the project.

## 5.6 Risk Analysis

During development of the DanceSoft project the project encountered a few possible risk that could effect the system. The team has found the following main risk within the system.

1. Interface Usability
2. Database Connectivity
3. Data Security
4. Data Backup

The first of the main risk is the PyQt interface usability. The system must maintain a simple and easy to use interface. This way the users can efficient and effectively access the system while maintaining the functionality at the systems core. The team must always keep this fact in mind when developing any of the systems pages and structures, as the users at the Academy are not assumed to have a technical background.

The second risk is the database connectivity. If the database can not be reached the project is incapable of accomplishing its functions. The system at its core is a database manipulation software, without the ability to connect to the database the project can not function. The system must therefore assure connection before allowing the users to continue.

The third set of risk with the DanceSoft system is database backups and data security. Database backups can be achieved by using functions within the MySQL database software. These backups can then be stored on a local system or wherever the client chooses to store the data. Database security can be achieved using prepared statements and queries, and other techniques of encryption and protection.

### 5.6.1 Risk Mitigation

The following section is the teams approaches, goals, and ideas to mitigate the various risk with the the DanceSoft project.

The teams has been developing each interface with the idea of usability in mind during the entire process. One of the testing approaches that the team used was user testing. This method meant sitting down and



acting like a user of the system and not a developer or technical user. If during this process something did not make sense or execution did not work in the expected way, the interface or function was reworked. Another part of this approach involved having people do sudo-beta test by sitting and attempting navigation of the system. If any questions or concerns came up from the user they were analyzed and addressed if necessary by the team. By keeping the interface simple and asking for feedback the team hopes to mitigate usability issues as much as possible.

The database risks are only partially dealt with in the current project. The database connection issues are handled through a few error checks in the connection. Also since the database will be contained in the local machine provided by the client the database can be connected to directly. This connection issue will increase once a student web interface is added to system, or if the client moves the database the connection will become reliance on the communication between either the two systems or the network connectivity at the Academy. Some of these future student interface issues were discussed before the student portal was dropped from the project. The risk could possibly be handled by storing the information the students are sending through a socket and storing it until the database is booted at the Academy. At which point the updates would be submitted. Another database risk is the security of the data stored within. The backup issue can be solved through writing the contents to a file which can be stored on either the local machine, or another machine as to protect the data should something happen to the computer in which the database is stored. Many MySQL management software also have ways of exporting the statements needed to create the database which the client can use to manage backups and create text file backups. The actual security of the data will not be completed in this iteration of the software. However the team has a few ideas for data security. Firstly since only the local system exist at this time, data should not be passed over a network. The SQL queries can be placed into prepared statement to aid in their security. It is highly advised by the development team that security solutions be explored by future developers as the current team lacks the experience and expertise to truly ensure the security of the data contain within the system.

## 5.7 Successes, Issues and Problems

### 5.7.1 Changes to the Backlog

Overall the project's base remained unchanged from the the beginning requirements. However as the project was being built the backlog and project encountered a few changes and iterations. These changes and modification are briefly explained in this section.

During the project the backlog was changed multiple times. The first of these redesigns occurred after sprint 2. The team and the client elaborated on some of the user stories to give the team an increased amount of focus for the various functions. This allowed the team to better develop the core system functionality required by the project.

However the first major change to the project's backlog occurred after sprint 3.5. It became clear to the team that the student web portal would not be completed by the end of the senior design project period. After discussions with the client in order to make the local desktop GUI as complete as possible the student interface was dropped from the project requirements. Users stories for this aspect of the project included: student registration, view tuition, and student schedules. In response to these changes the team ported the functions into the local desktop interface to create the student registration, billing, schedule, and other interfaces.

The last changes to the backlog occurred in the last sprint of the project. The project had been refined at this point as a first iteration of the DanceSoft project. This change became necessary as issues arose within the team that prevented the team from reaching full project completion. As a consequence of this the project became a proof of concept or minimum viable product for the client. The team demoed the existing software for the client, at this meeting the client and the team discussed the requirements again and reevaluated the project requirements for completion, and the senior design fair presentation.



## 6

---

## Prototypes

---

This chapter a log of the various stages of the DanceSoft prototype. Each prototype is organized by the South Dakota School of Mines Senior Design class sprints.

### 6.1 Sprint 1 Prototype

The goal of the first sprint was to collect the requirements for the project. Once requirements gathering was complete the team moved to a research phase with the end goal of deciding on how the three main parts of the project would be implemented. These three parts were the programming language, the user interface framework, and the database type. Through the research the team came to three decisions.

The first of these was the programming language. The team choose Python as this iterations programming language. Python has the upside of being a language the team is more experienced in. Also the client Dr. McGough knows Python so any updates would be easier for him to work on should he decide to work on the project after this team is disband. Python is also a cross platform language which allowed the team to produce working code for Mac, Windows, and Linux at the same time. This means the the team can develop on Windows, a development environment we are more familiar with overall. Also development in Python means that if the Academy was ever sold to a Windows user the product would still work. Another advantage we discovered to python is the academic and career experience it provide since in our research for the SD Mines career fair we discovered that many companies use Python and more than expected would like QT experience. Lastly the Python community is larger and can more readily provide assistance if needed through websites and research. This does not mean that other languages like Swift were not viable choice but the time spent learning a completely new language and it's GUI toolkit would have added significantly to the project learning curve and reduce even further the amount the team could have completed.

The second part of the research was the GUI toolkit. After looking into GUI kits like Kivy and PyQt the team decided to go with PyQt as Kivy would have added to much to the ramp up time. Also PyQt provide easy to use tools for GUI design through things like the design, and more help within the community.

The last research was in database type. The database aims to store employees and students and classes information and the size of students less than 4000 and the size of employees less than 10. The structure of data is stable, by considering all those facts above, we decide to use SQL database. For the type of MySQL database, the team thought of using MySQL. MySQL is a free software and widely used in different fields. It has very good portability and can run over different OS systems and aims for small size of data. Also our experience is more focused in MySQL as a part of the South Dakota School of Mines and Technology, and the feature set for SQL in MySQL fits within the scope of the project, for these reasons we choose MySQL as the database frame work.

The last part of the prototype for this sprint was learning the materials we had selected. The team began practice with the PyQt tools and developed a sample window to learn the basic things available for the system.

Figure 6.1: Tables currently in database

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Data
Account	MyISAM	10	Dynamic	3	30	112.0 bytes	256.0 TB	5.0 KB	5.0 KB
Address	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Admin	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Class	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Guardian	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Student	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Student_Class	MyISAM	10	Fixed	0	0	0.0 bytes	4096.0 TB	1.0 KB	1.0 KB
Teacher	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB
Teacher_Class	MyISAM	10	Fixed	0	0	0.0 bytes	256.0 TB	1.0 KB	1.0 KB

### 6.1.1 Deliverable

1. The research into program languages, database and GUI frameworks and architectures for the DanceSoft project.
2. Final decision on frameworks and architectures for the DanceSoft project.
3. User Stories and Product Backlog for the project.
4. Creating a simple Qt window.

### 6.1.2 Backlog

1. Set up Github (source control)
2. Design Research
3. Design Decision
4. Create first window
5. Sprint 1 documentation
6. Sprint 1 Review
7. Continuing practice with QT

### 6.1.3 Success/Fail

The team was able to make framework decisions and begin to learn the toolkits for use in the project. Overall this sprint was a success.

## 6.2 Sprint 2 Prototype

As of now the prototype is in a state to begin working on functionality. The rough (not final) GUI pages are laid out and being constructed to give us an environment for creating the functionality. The back end database has been constructed and will allow us to begin testing the database connected page as we create them. Current pages constructed are log in page, landing pages, and some options pages.

The first thing we tackled in this prototype was getting the database up so we could begin actually developing the product and give our qt interface something to actually interface with. The main goal here was to make sure we had constructed the database in such a way that it could complete all the user stories it needed to in a way that made sense. After talking through the tables and the users stories we came up with the table creation script.

The GUI pages for this sprint were the log in page and the landing pages for admin and teachers. The first page the team constructed was a log in in page that takes a user name and a password and first checks to see if they exist and are correct within the database. If they are not a dialog saying whats wrong appears

Figure 6.2: Current iteration of the log in page

and prompts the user again. If the information passes the check the system reads the users permission level and sends them to the corresponding landing page.

The other set of pages we need to make to begin writing functionality was the landing pages for the admin and teacher permission levels. These pages show the types of things each can do and allow navigation to the various different functionality. For example the admin landing page has a button to take the user to a student options page, from there the user can select which option they want and will be taken to the page to execute that function.

### 6.2.1 Deliverable

1. The database creation script
2. GUI path work
3. Log in page that reads users permission level and sends them to the correct landing page
4. Rough landing pages for Admin and Teachers (design improvements to come in later sprint)

### 6.2.2 Backlog

1. Creation of database tables
2. Draw GUI path
3. Create a log in page that sends the user to the correct landing page based on their permission level
4. Create versions of teacher and admin landing pages to test functionally (improve look in latter sprint)
5. Continue GUI page creation to have environments for functionality
6. Sprint 2 Review
7. Continuing practice with QT

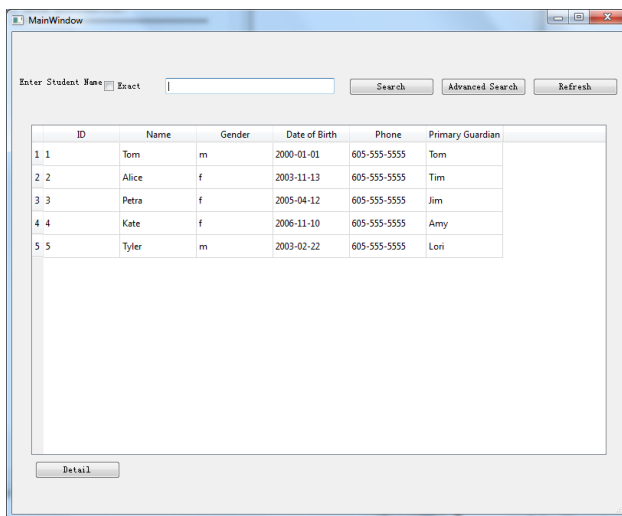
### 6.2.3 Success/Fail

The team was able to produce the log in page and the landing pages for both admins and teachers. The team was also able to create the database on the School of Mines server and get the general structure of the tables up and running. The team did begin to experience issues with time owing to the loaded academic commitment of both team members. Also causing issues was the teams inexperience with PyQt and each other. The team communication was flawed in its execution and inexperience on the part of both team members with the GUI caused further slow down.

## 6.3 Sprint 3 Prototype

The progress is being made on the simpler functions of the projects including search, updates, and role sheets. As of now the prototype is in a state of some working functionality on the admin and teacher side. The pages are not yet tied together but the complete functions function independently. Several GUI pages are complete or in a working state to compliment these functions. The back end database has been slightly updated to adjust to the needs of the project. Lastly the prototype has reached a state where testing can be conducted on parts of the project.

The first page tackled was the search page. The page takes user input and searches based on name using a fuzzy search. Also included in both searches is an advanced search option that allows the user to check boxes corresponding to the fields in the database, which in turn allows the user to search in more versatile ways. Lastly the user can click on a student to pull up all of their information and modify it as needed.



Another page prototype is the "Add a Class" page, by clicking on the "Add a Class" button on the admin landing page will open a dialog. From this dialog box the user can type in information to add a class to the database. The role sheet page generates a list of classes teachers are currently teaching. Teachers can then see which students are enrolled to take his or her class by selecting the corresponding class on the list. If the user wants they can print this list as a pdf.

Further pages added during this sprint were the update pages. These pages are fairly simple to explain, sometimes the users of the system will need to add or alter a record of some kind these pages need to be able to do so in a simple and concise way. The update pages worked on in this sprint include student information from the employee side, updating teacher information and updates to clothing requirements for a class. Most of these page will just produce a form where information can be displayed and updated.

The assign teacher to class page allows an admin to select a class, clicking on a class pulls up a list of available teachers to select to teach the class. Clicking on a teacher will pull up a message box to confirm the selections before submitting to the database. The available teachers are selected based on the start time of the selected class and the end time of the classes already being taught by each teacher. If the selected class is already assigned the user is prompted with a dialog where the user can select whether or not they would like to reassign the class to a different teacher.

### 6.3.1 Deliverable

1. Student search page
2. Employee search page
3. Add a class to the database
4. Update teacher and student information pages

5. Modify clothing requirements for a class
6. Generate a class role sheet
7. Assign teacher to a class
8. Ability to add and subtract students from a class

### 6.3.2 Backlog

1. Queries in database to retrieve student and employee information
2. Query and insert information needed to create and new class and produce a results page.
3. Query database to produce class role sheet
4. Create clothing requirement update function, and add clothing function for teachers and admin
5. Create update query to assign teachers to a class
6. Create ability for employee to look up and modify student information
7. Add and subtract students from a class
8. Sprint 3 Review
9. Create Client presentation

### 6.3.3 Success/Fail

The team was able to complete several main pages contents to various user stories. However the weight of other academics slowed the teams progress and of course the with the knowledge of the tools growing it was less of a problem then before but still persisted.

## 6.4 Sprint 3.5 and 4 Prototype

During sprint 3.5 the team organized the functionality and tied them together to create the single current working prototype. Other updates were made to fix known bugs and add a class search function. While the team worked on putting the project together smaller bugs or needed pages that were implied by user stories were tackled as they came up. This resulted in too much time being used, so the student interface was not tackled. The student interface has since been dropped from the clients project requirements due to time.

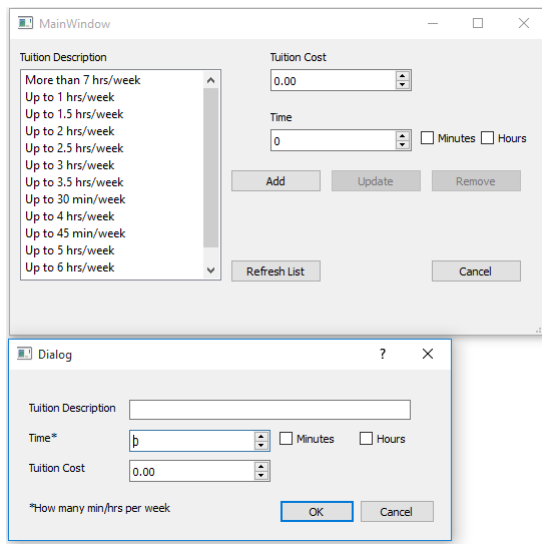
Sprint 4 was dedicated to payroll and billing. First the client rewrote the user stories to provide more clarity to the team. After this the user stories were tackled by the group to produce the first draft of the payroll and billing interface. This included logging hours, payments, fees, rates, and credits. After this sprint the team demoed the project for the client and got notes on the project as a whole. These notes will be tackled as part of the backlog in the next sprint.

**Prorated Refunds:** The prorated refunds code will keep track of how much a class costed a student. If the student drops a class it will refund the remaining worth of the class to the students school credit field. This file is still in progress at the time of this prototype.

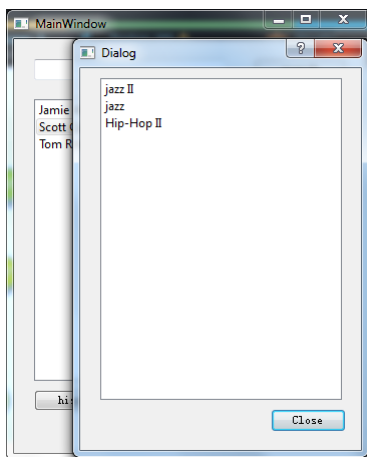
**Enter Staff Hours:** This page allows an admin to log teacher hours and add different pay rates such as driving rate to the system. There is a default class rate for all teachers. This is further address in the success/fail section below.

**Enter Tuition Rates and Fees:** These pages allow the user to look at the current tuition rates and fees. The user can then update existing rates, add new rates, or remove rates that are no longer needed. Tuition rates

are logged in the database as flat minute rates. The user can enter new rates in the form of minutes or hours.

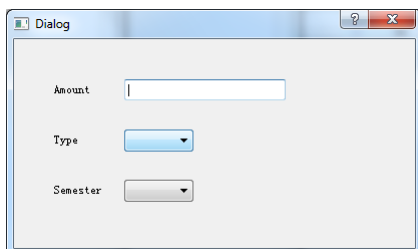


**Teacher History:** This page includes several components. Firstly, user can use this page to find a specific teacher by entering a full or partial name of teachers. Secondly, users can click the history button to open up another window which has a list of classes the teacher has taught.



**Set Semester:** This page helps users to set the current semester from pool of semester and add a new semester to the system.

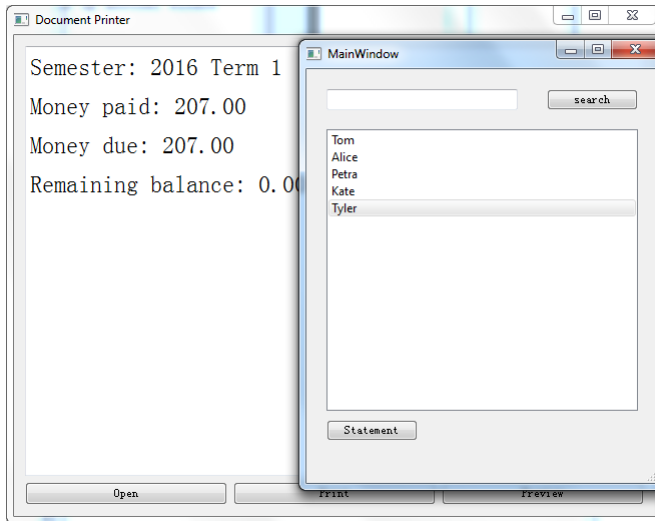
**Partial Payment** This page includes several components. Firstly, admin users can use this page to find a specific student by entering a full or partial name of a student into the search bar. Secondly, users can click the pay button to open up another window which asks the user to input amount of money paid, payment methods and semester paid. The user also can choose single or multiple students at one time.



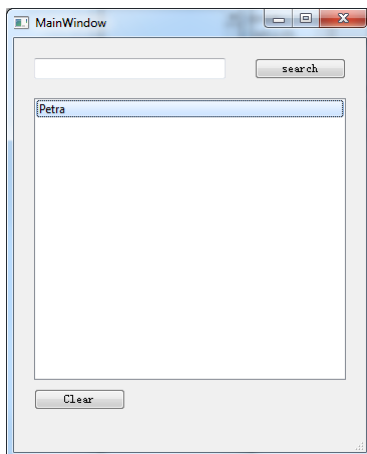
**Student's owe:** This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the statement button to open up another window which shows the amount of students paid, the amount of students due and the



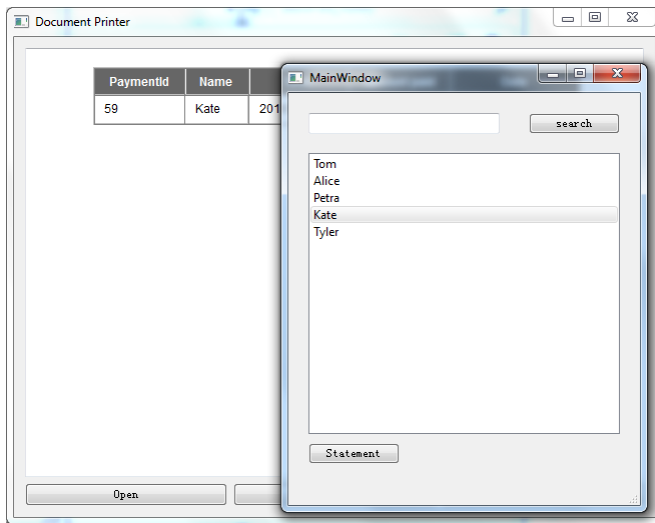
balance of student's account. Users also can print the invoice generated by the window.



**Enter Payment:** This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the clear button to clear the due of selected students. Users can select single or multiple students at one time.



**Bill History:** Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the statement button in order to get the list of payments. In addition, user can print the bill history.



### 6.4.1 Deliverable

Sprint 3.5 had no defined deliverables, and was mostly clean up of user interfaces and functions that were not directly requests by the client but were implied by other client request.

Sprint 4 deliverable was the first draft of the payroll and billing interfaces and functions for the project.

### 6.4.2 Backlog

Sprint 3.5 Backlog:

1. Fix Assign Teacher to Class Dialog Bug
2. Add Class Search
3. Complete Role Sheet
4. Tie Functions Together
5. Fix Bugs

Sprint 4 Backlog:

1. View Teaching History
2. Look at The Current Amount Someone Owes
3. Generate an Invoice For The Amount Due
4. Look at Billing History
5. Apply Credits to a Students Account
6. Enter the Tuition and Fees Rate
7. Give Early Registration Discounts
8. Enter Staff Pay Rates
9. Enter a Full Payment for One Student
10. Enter a Full Payment for Several Students
11. Enter Payments From Multiple Sources For One or More Students

12. Compute Teacher Wages
13. Enter Staff Hours
14. Give Prorated Refunds

### 6.4.3 Success/Fail

During this sprint the team was able to finish most of the rudimentary billing prototypes during these sprints. Most of the prototyping was spent fixing bugs, adding implied functions and making updates.

During this phase the team realized that the project would not reach completion by the end of the project. So the team met with the client to discuss the student interface. After the meeting it was determined that the team could not complete the student interface in time and if it was attempted it would prevent the completion of the local interface as well. As a side affect of this change the team dropped the accept/remove student function was dropped since the function was meant to approve student web registration, then a local student registration function was added to the local GUI to compensate.

## 6.5 Sprint 5 Prototype

Sprint five was dedicated to finishing the last of the functions and clean up of the project. First the client modified some requirements for the team. After this the user stories remaining from sprint four were tackled by the group to produce the remaining payroll and billing interface. This included logging hours, and payments. After this was done the team started on bug fixes and quality of life updates such as new buttons and removal functions. The team is currently working on finishing up the student registration. If progress continues at this rate the prototype should be completed functionally for our teams iteration however without some work the prototype may not reach some of the teams desired standards.

During sprint five the team tackled the remaining functionality, bugs, and quality updates in an attempt to finish the base project. Removal functions to clean out the database were added, and some client requested quality of life updates were added for this project version. Bugs and glitches were patched up in many functions. Another job of this sprint was to finish the rollover from sprint four since some payroll functions still needed work. Lastly the team tackled student registration since the student interface was dropped from the project requirements in sprint 4. The registration is still in progress at this time, as the team failed to complete this functionality in time for the end of the sprint. However it was completed during final sprint.

**Prorated Refunds:** The prorated refunds code will keep track of how much a class costed a student. If the student drops a class it will refund the remaining worth of the class to the students school credit field. This file has now been completed but due to client request and Academy no refund policy, the team does not foresee this function being useful to the system since the credits can store this information. However this function is left in the project/proof of concept at the request of the client Dr. Jeff McGough.

**Enter Staff Hours:** This page needed to undergo changes after meeting with client. The new version will allow a teacher to log their in class hours, office hours, trip hours, etc. These hours will then be paid out at their different rates respectfully. The hours will be logged as single numbers not by days or weeks. This function is now completed, as of sprint 5. However as of sprint 6 this function has been added to the rudimentary billing prototype however the team did not add different course rates for each teacher.

**Enter Teacher Wages:** This was handled in the database in previous sprints assuming one pay rate. However in talks with the client the academy pays different pay rates based on what part of academy work they are doing. So the page will need to be updated once the list of possible pay rates have been provided by the client. This pages and its updates are still in progress. Like the enter staff hours page the course hour rate is the same among all teachers in this version, this will need to be updated in a future iterations of this project as this team was development frozen before changes could be made.

Dialog

Payname  Payrate

rr  8.01

Add Update

Enter Tuition Rates and Fees: Updated these pages to provide quality of life updates to the function requested by the client.

Dialog

Description  Annual Discount Percent

Cost  0.10 ☐ Flat Rate ☐ Percent

OK Cancel

Bug Fixes: During the course of testing and the client meeting some bugs or inconsistencies were discovered within the project. The bugs found and fixed are:

1. Student search was not showing all the students. Status: Fixed
2. Search edit need to be turned off and advance search modifications. Status: Fixed
3. Search, role, and schedules need to have dynamic not static times. Status: Fixed
4. Allow for time changes on schedule. Status: Fixed
5. Some of the back buttons in the teacher interface did not work correctly. Status: Fixed

Updates Crossover: One of the updates that needed to be completed this sprint was the need for admin and teacher updates to crossover if something was changed. This means that if someone is an admin and they update their phone number through the "update my information form" then the phone number will be updated in both the teacher table and the admin table. This way the system will not have conflicting information in different places.

Approved/Rejected Student Updates: A update requested by the client was the ability to see rejected students in the approved/rejected pages just in case the academy changed its mind about a student. The page underwent slight modifications to the way information was displayed to make this change efficiently.

Further updates were made in sprint 6, because this function was meant to deal with student approval from the web portal, however since the student interface was dropped from the requirements this function no longer serves a real purpose within the system. If the future when the student portal is added this function may be useful again, as of now the function is left in the project at the request of the client.

MainWindow

Hip-Hop II  
jazz  
jazz II  
Tap I  
Tap II  
test  
testing

	Student_name	Class_approval
1	Alice	1
2	Kate	1
3	Petra	-1
4	Tom	-1
5	Tyler Cole	-1

Remove Add

\* Class Approval: Pending: 0 Approved: 1 Rejected: -1

**Admin List:** In creating some of the updates it became clear that a specific admin list was needed to see who had admin access to the system. This was done through a simple list view of the names that can then be selected to display more detailed information about the selected admin.

**Add/Remove Locations:** Another requested feature was the ability to add and remove locations for classes in the database. This feature is necessary because the Academy sometimes teaches classes in different places based on need. The team accomplished this using a simple dialog that connects to the location field of the class table in the database. When the user updates or adds a new class the location combo box in the form is updated. The user can then select the new location and proceed to finish with class information.

**Removal Functions:** These are simple functions that pull up a dialog box where the user can select a name and remove all the information related to that user from the system. The removal functions that exist in the system are: teacher, admin, student, and class. Teacher will remove that persons information from the system. Admin removes that persons admin rights and then ask if the whole user should be removed. Student removes all the student information including data, schedules, guardians and addresses. Users should be careful when running any of these functions as the removals are complete and can not be undone or recovered.

### 6.5.1 Deliverable

Sprint 5 deliverables were the remaining functionality for the redefined requirements from the client for the project. If the sprint was successful the main project will be done or very close to done functionally for this iteration at the end of the sprint.

### 6.5.2 Backlog

1. Admin and teacher update crossover
2. Ignoring case in address forms
3. Add rejected students to approved/ reject and provide current status
4. Multiple pay rates

5. Admin list function
6. Enter staff hours
7. Sprint 4 rollover (remaining payroll functions)
8. Add and remove class location
9. Removal Functions (Admin, Class, Student, Teacher)
10. Refunds and check boxes for fees and tuition
11. Quality of life updates to some functions
12. Bug Fixes
13. Student registration
14. Employee wages
15. Begin User Guide for documentation

### 6.5.3 Success/Fail

At the end of this sprint the project development has been frozen by the client. The project was updated and put together in preparation for South Dakota School of Mines Design fair. As of this final prototype the system has provided the proof on concept now requested by the client. The team will now hand over what was completed to the client along with this document, and the client will decide what to do with the prototype from there.

Since the team was unable to produce a fully functioning project, the project was redefined as a proof of concept for the system and the future of the project will be decided by the client at a later juncture. Since the senior design class has ended and the current DanceSoft team is separating to moving to other projects and jobs.

---

## Release – Setup – Deployment

---

This section contains a list of the deployment dependencies, deployment instruction for future iteration development, development setup instructions, and a brief description of the concept and future versions and development of the DanceSoft product.

### 7.1 Deployment Information and Dependencies

The DanceSoft project created by this team is meant to be a proof of concept for the creation of a administrative software for the Academy of Dance Arts. Since this is a first iteration of the software, the system is not meant to be deployed for industrial use at the Academy. The project contains several install dependencies since a actually installer was not developed for this project.

In order to run this iteration of the DanceSoft software the user will need to install several separate components, these include:

1. A valid instillation of Python 3 which can be acquired from <https://www.python.org/downloads/>
2. The PyQt 4 libraries and the PyQt designer suite acquired from url<https://www.riverbankcomputing.com/software/pyqt>  
The product will not work with PyQt 5 without modification due to the incompatibilities between PyQt 4 & 5
3. A MySQL database that can be used to store data, and possibly a database management software like MySQL Workbench for direct data manipulation and continued development. You can download MySQL free of charge from <http://dev.mysql.com/downloads/> and MySQL workbench can be found at <http://dev.mysql.com/downloads/workbench>. However any MySQL database manipulation software should work with the project, so the user can use their preferred software.

Once these main three software packages or libraries are installed the user should be able to open all the Python scripts for development or run the "login.py" script to run the software. Other Dependencies for this project include:

1. The database connection - If the software can not connect to the database either over a network or a local machine, the product will be rendered useless on account of the log in page and most to all pages relying on the database to function correctly.
2. The PyQt 4 libraries - Like with most programming languages and libraries if the standard functions are modified it can break or alter the software functionality. However this should not be a problem when deploying the system due to the main work being done on PyQt 5.
3. PyQt installation - During Deployment if the PyQt libraries are installed incorrectly then the system will fail to deploy/run effectively.
4. MySQL
5. Python 3

## 7.2 Setup Information

Setup for further development of the DanceSoft has several steps and pieces as no installer was developed within the time frame of this iteration. The following is a general overview of the steps to step up for development of the system.

Steps for running the software:

1. Install Python 3
2. Install PyQt 4
3. Install MySQL
4. Download the DanceSoft python scripts
5. Run the login.py script

### 7.2.1 Installing Python 3

The first major component of the system is the main programming language for this version, Python. Python has two main versions at the time of writing this document, these are Python 2.7 and Python 3.4, the Dancesoft project was developed using python 3.4 and therefore will not work on Python 2.7. To install Python 3 go to <https://www.python.org/downloads/> from this site a developer can download several version installers for Windows, Mac OS X and Linux/UNIX. Once the installer has been run a developer can confirm success by running the Python 3 command from the command line.

### 7.2.2 Installing PyQt 4 and Designer toolkit

A developer can install the PyQt 4 library from <https://www.riverbankcomputing.com/software/pyqt/download>. The PyQt 4 library can also be copied from the DanceSoft git repo located at <https://github.com/SDSMT-CSC464-F15/dancesoft>.

The riverbank website contains installers for the PyQt library that include all needed components for the system. On Mac or Linux user can download the snips or source code from the website, or use one of the many tutorials online if needed. The team recommends just copying the files from the repo and placing the PyQt folder in the site-packages sub-folder of the Python 3 folder.

### 7.2.3 Installing MySQL

The database component used by the team was MySQL, this software can be download and installed free of charge from <http://dev.mysql.com/downloads/>. Further manipulation software such as MySQL Workbench, or Navicat can be installed to help with development. Several pieces of software can also be found on the MySQL website, however any MySQL development software can be used. It is up to future developers to find the software that fits their needs the best.

### 7.2.4 Running Project

Once all the necessary components have been downloaded and installed a developer can write/run the Python and PyQt scripts for the project. To start the current system the user must run the "login" script to pull up the log in window and begin using the software.

At the time of project submission the project contained an admin log in using the user name: jwitmore and the password: redance. This account was used both for testing and client demo purposes during the initial program development.



## 7.3 System Version Information

Past version of this system have been presented to the client as senior design projects, however due to reasons unknown by this team the projects were abandoned by the client. Therefore the previous versions have no effect or input on this teams proof of concept for the DanceSoft system.

The system is in a pre-industrial version noted in this document as version 1.x.x this iteration of the DanceSoft project is a proof of concept presented to the client Jeff McGough. The idea of this first iteration is to show that a true in use launch of this software is possible. Whether this is through a follow up senior design team that will use this teams iteration as a blueprint, Dr. Jeff McGough continuing development himself, or finishing the project through a contractor, or industrially mentored senior design team.

This version shows execution of the main functions laid out in the initial requirements. These functions provide an ideas for a minimum viable software. The first version of the software was meant to have a full interface and a student web portal for student registration, billing, and information management. However due to time constraints, DanceSoft team issues, and to many other academic commitments from the two members of the DanceSoft team, the client converted the projects goals from a industrial launched product to a proof of concept for future iteration and versions.

Future versions of the software however the client may choose to create them will most likely include more refined functionality, the addition of the student web portal if possible, more database concurrency, and security. Even though the current version is a proof of concept as described in the DanceSoft software contract; It is the belief of this team that the final version of this software is possible through future iterations, and versions.

## 7.4 Future Work

The following is the remaining work that needs to be done on this project should this iteration be directly continued. If the project proceeds using different tools or to a contract job, this section can be ignored.

1. The project and database are currently not secure, security will need to be implemented in the system.
2. The student web portal
3. The connections in the local GUI to the student web portal
4. The billing and payroll functions are currently very rough, and will most likely be redone in the future.
5. Importing and Exporting data to the database
6. Database backups
7. A different class hours rate for each teacher since all employees are not paid the same for class time.
8. Refunds and an old version of the Add/Drop students where left in at the request of the client, but overall serves no purpose within the system.
9. Fees are listed in the system however they do not currently do anything else within the system
10. Family billing and setting discounts where not implemented in this version of the system.



## 8

---

# User Documentation

---

## 8.1 User Guide

The following is the user guide for the DanceSoft project:

### 8.1.1 Entering the Software

Login page:

When the user first logs into the system the user is prompted with a log in window where they can enter their user name or pass word for the system. If the user enters the wrong password then the system pops up a dialog and tells the user to reenter there password. The default user name for a new user is the name used to enter the system. So if I enter a new teacher named Marcus Berger then the default user name is Marcus Berger. The default password is redance, both the user name and password can be changed using the change functions in the personal section of the teacher landing page.

Landing Selection:

After the user logs in they can select which of the two landing pages, Admin or Teacher that they want to go to. If the user does not have admin permissions the user will be blocked from going to the Admin Landing page.

### 8.1.2 Admin Landing

Once an admin has logged in they can select the admin landing page to be taken to a variety of functions listed below. In the main page the user has four options for type of functions to choose from employee, student, class, and billing.

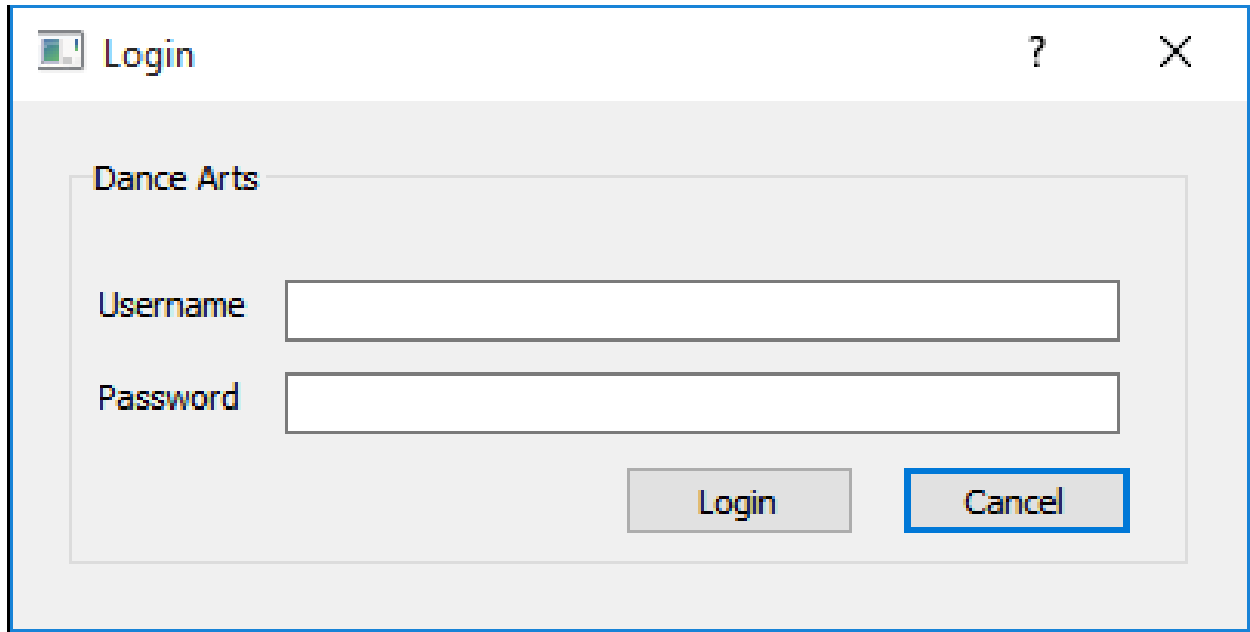
#### 8.1.2.a Manage Employees

When the user clicks on manage employee they are given a set of buttons that they can choose from.

Search Teacher:

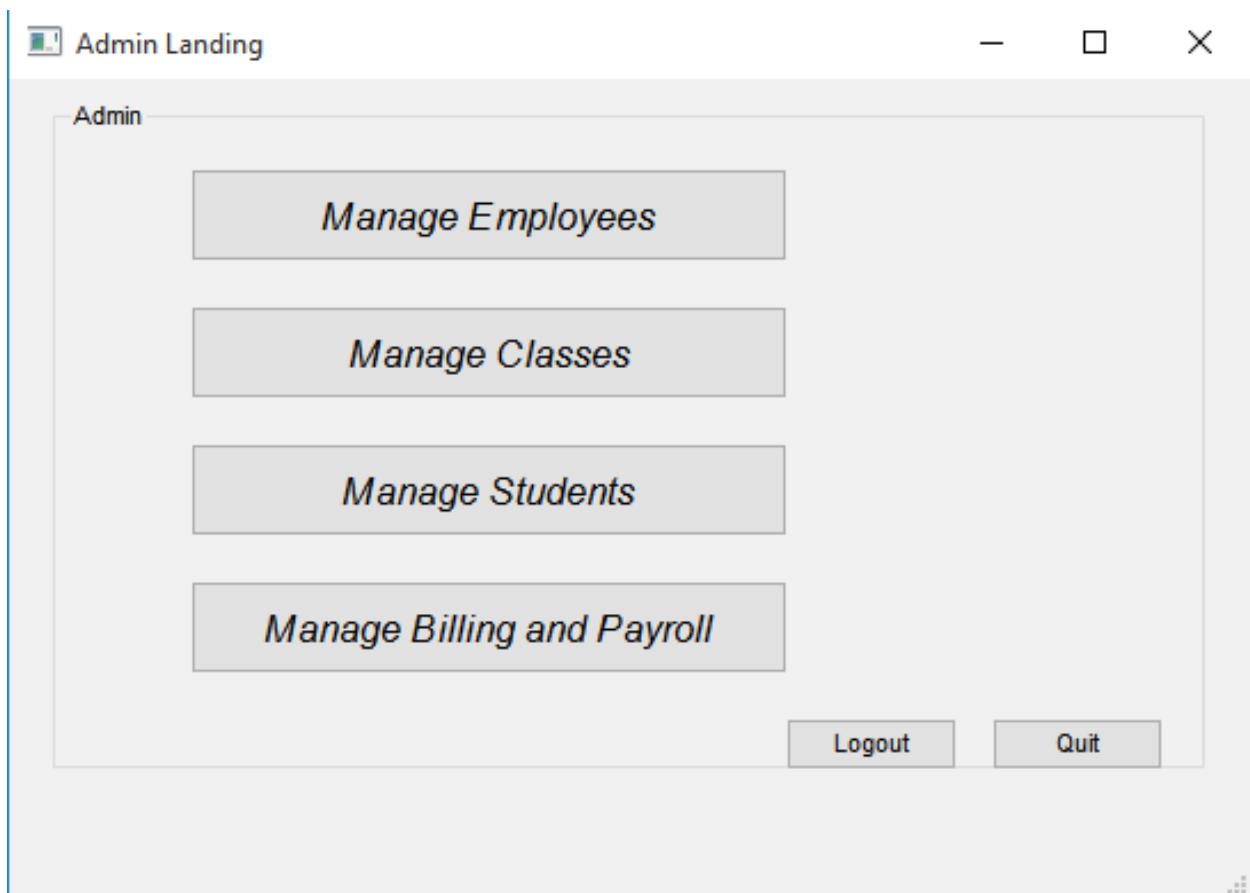
When the user selects search teacher, the system will bring up a window that displays the teachers in the system. The user can then type a teacher's name into the search bar and the field will display the teachers that names contain the entered value. If the user is looking for an exact match then they can check the "exact check box. After the user conducts a search, they can click the refresh button to bring back the full list of teachers in the database.

By clicking the advanced search button the user can further refine the search value, the advanced search dialog is accessed by clicking the advanced search button. Advanced search allows the user to to search for teachers by id, name, phone numbers, or date of birth.



A screenshot of a 'Login' dialog box. The title bar shows a small icon, the text 'Login', and standard window controls (help, close). The main area is titled 'Dance Arts' and contains two input fields: 'Username' and 'Password'. Below these fields are two buttons: 'Login' and 'Cancel'. The 'Cancel' button is highlighted with a blue border.

Figure 8.1: Log in page



A screenshot of the 'Admin Landing' page. The title bar shows a small icon, the text 'Admin Landing', and standard window controls (minimize, maximize, close). The main area is titled 'Admin' and contains four large buttons stacked vertically: 'Manage Employees', 'Manage Classes', 'Manage Students', and 'Manage Billing and Payroll'. At the bottom right of the main area are two buttons: 'Logout' and 'Quit'.

Figure 8.2: Admin Main Page

Search

Enter Teacher Name ☐ Exact

	ID	Name	Home Phone	Cell Phone	Work Phone	Gender	Teacher_hours
1	1	Jamie Witmore	605-430-8463	605-430-8463	605-392-5566	Female	2.2
2	2	Scott Carda	605-394-4444	605-394-4300		Male	1
3	3	Tom Robinsen	605-555-5555	605-555-5555	605-555-5555	Male	2
4	0	None	None				

Figure 8.3: Teacher Search Window

Once the user's desired teacher is found, the user can click on the name in the main window and select a details button to view all the information for that teacher. In the details window, the user is able to change the information for an entry and submit the updates to the system.

#### Show Admin List:

On the main admin page, the user can select the admin list button. This will display a text list of admins for the system. The window contains a search bar to refine the list if the user is looking for a specific entry. The page contains a details button which will allow the user to view the full details of the entry. The add button allows the user to select a teacher in the system and give them admin permissions. The remove button allows the user to remove admin permissions from the system; this, however, does not remove the teacher from the system.

#### Update Teacher Information:

The update teacher button on the admin employee page allows the user to select a teacher in the system and populate a form. The user can then change any of the fields and submit the updates to the system's database.

#### Enter Teacher Hours:

Select Teacher	Jamie Witmore ▼
	Select
Name*	Jamie Witmore
Home Phone*	605-777-7777
Cell Phone	605-777-7777
Work Phone	605-655-5566
Address*	333 Elm Street
City*	Rapid City
State*	South Dakota ▼
Email	jamie.witmore@rcdancearts.com
Gender	Female ▼
SSN*	444-44-4444
Pay Rate	11.85 ▲▼
Medical Information	Allergy to seafood
* Required Field	Submit

Figure 8.4: Update Teacher Form

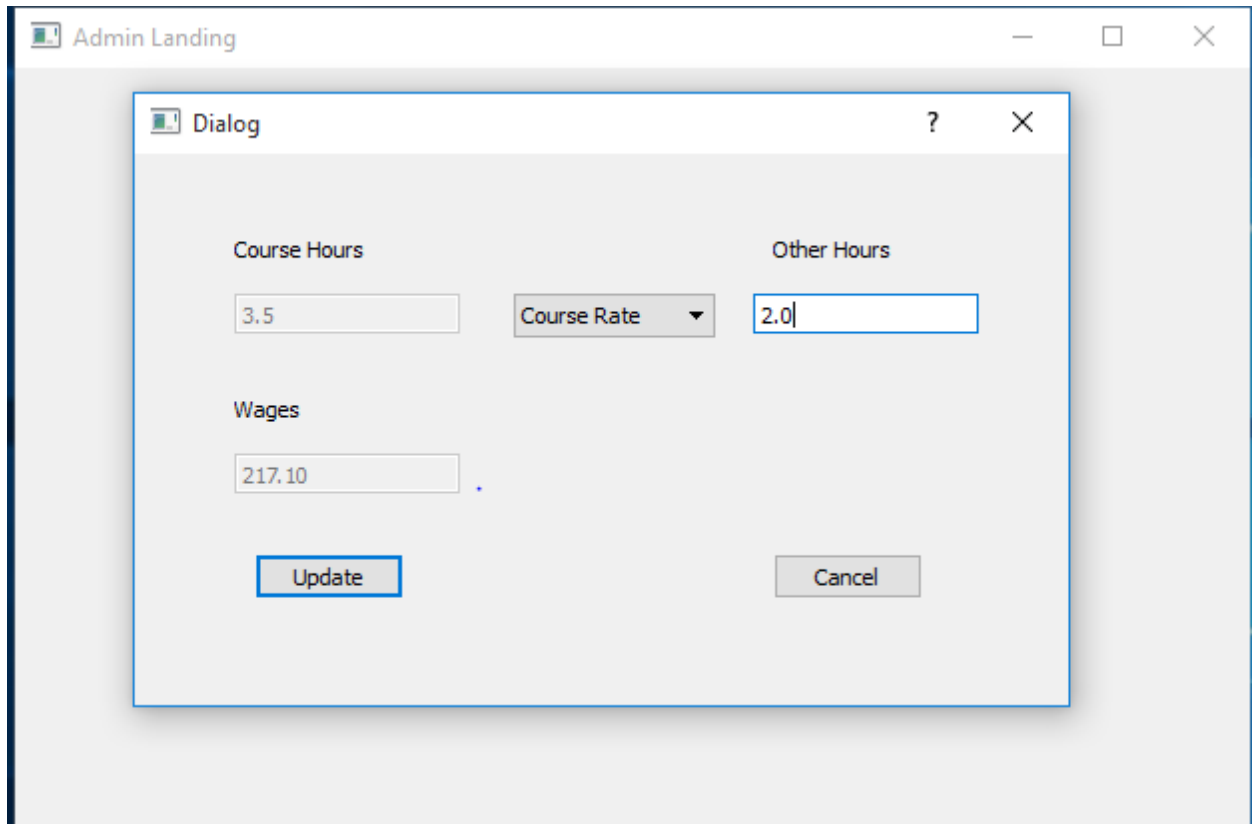
The image shows a screenshot of a web application interface. In the background, there is a window titled "Admin Landing" with standard window controls (minimize, maximize, close). Overlaid on top of this is a smaller dialog box titled "Dialog". The dialog box has a light gray background and contains several input fields and buttons. At the top left of the dialog is a label "Course Hours" above a text input field containing the value "3.5". To the right of this is a dropdown menu labeled "Course Rate" with a downward arrow. Further right is a label "Other Hours" above a text input field containing the value "2.0". Below these is a label "Wages" above a text input field containing the value "217.10". At the bottom left of the dialog is a blue button labeled "Update", and at the bottom right is a gray button labeled "Cancel".

Figure 8.5: The Admin Hours Page

Another button on the admin landing page is "Enter Teacher Hours". The user is able to select a teacher and a dialog pops up. The user is then able to view the selected teachers hours in each pay rate and change them if the user so chooses. The gross wage is then calculated and displayed to the user.

#### Assign Teacher to Class:

This button brings up the assign teacher window. From here the user is able to select a class name, if the class is currently assigned to a teacher a dialog box will pop up asking the user if they would like to reassign the class to a different teacher. If the user selects yes or if the selected class is currently not assigned to a teacher a list of teachers available at the class time will pop up and the user can select a teacher to assign to the class.

#### Teaching History:

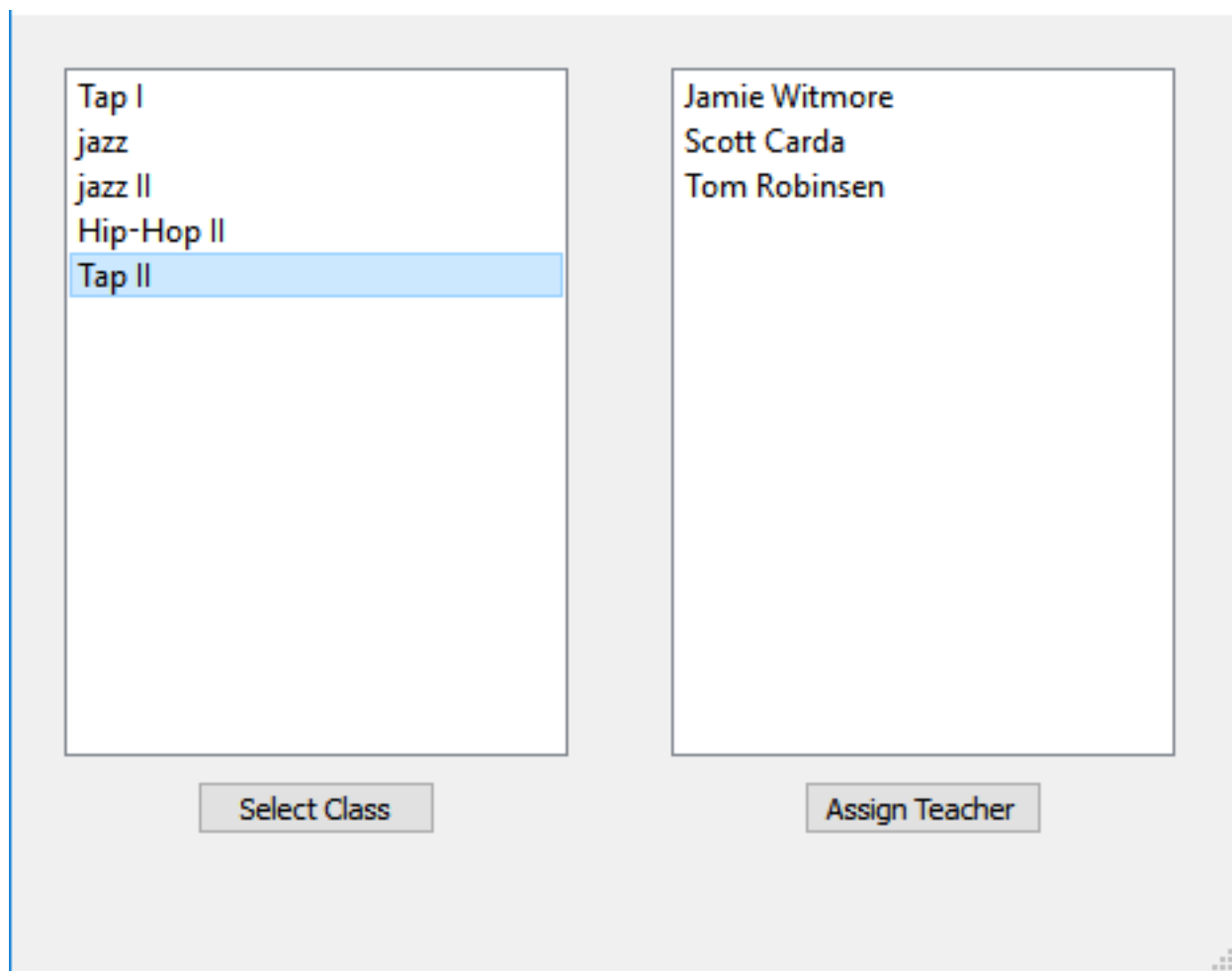
The user is able to select a teacher and press the history button, this brings up a list of all the classes that teacher has taught, excluding the current semesters classes.

#### Enter New Teacher:

This button brings up a form where the user can enter the fields for a new teacher. Once all the required fields are filled out the user can submit the teacher to the system. The required fields are name, date of birth, home phone, address fields, gender, and SSN. When the submit button is clicked, the user will be able to review the entry before submitting it to the system.

#### Remove Teacher

The user is able to select the name of a teacher and remove the selected teacher from the system. This removal is complete, this means that if the user removes a teacher it will completely remove all the data from the system connected to that teacher. This includes classes, teacher data, and hours.



The image shows a software window titled "Assign Class Window". It is divided into two main sections. The left section contains a list of class names: "Tap I", "jazz", "jazz II", "Hip-Hop II", and "Tap II". The "Tap II" item is highlighted with a blue background. Below this list is a button labeled "Select Class". The right section contains a list of names: "Jamie Witmore", "Scott Carda", and "Tom Robinsen". Below this list is a button labeled "Assign Teacher". The entire window has a light gray background and a blue border.

Class	Teacher
Tap I	
jazz	
jazz II	
Hip-Hop II	
Tap II	

Figure 8.6: Assign Class Window



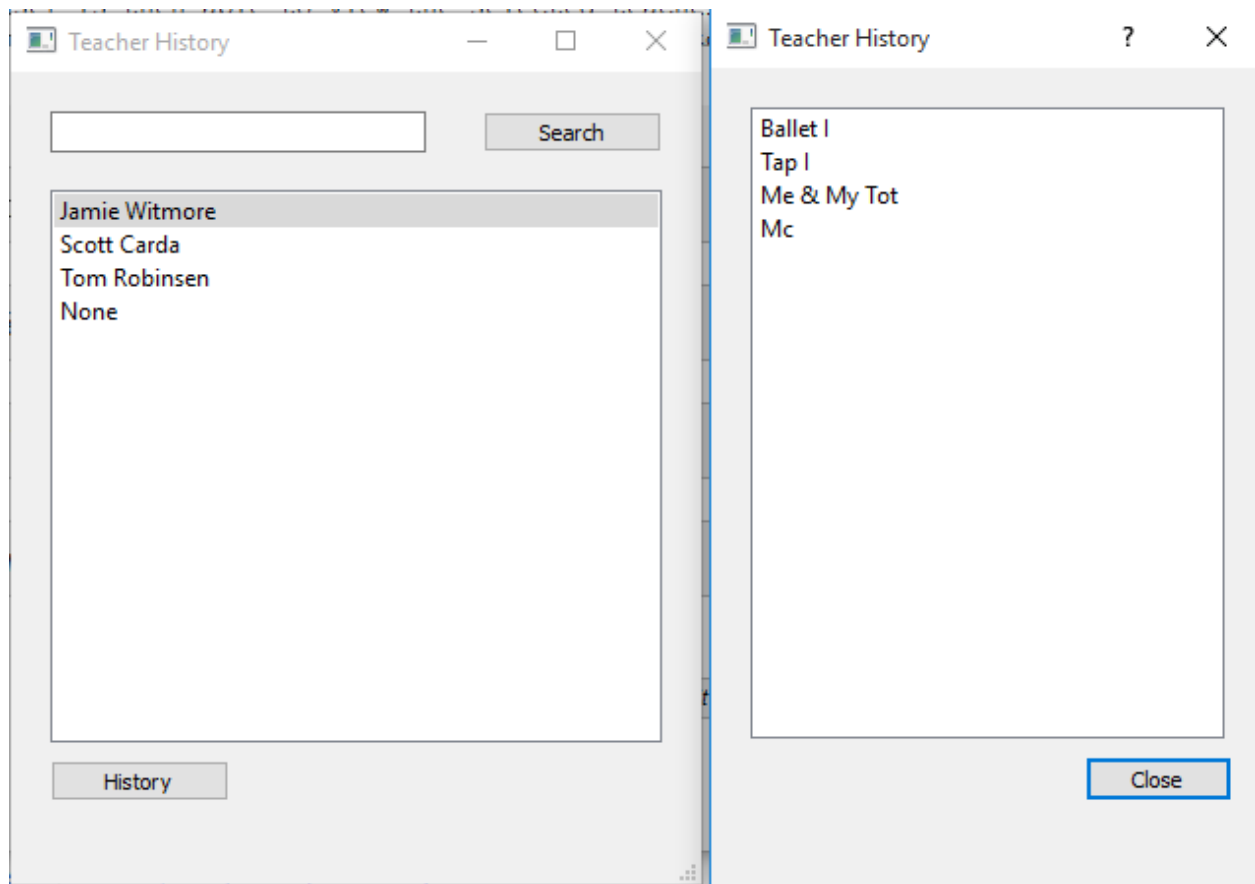


Figure 8.7: Teacher History

Class Search

Enter Class Name ☐ Exact

	ID	Name	Start Time	End Time	Class_day	Location	Capacity
1	1	Tap I	3:30:00 PM	4:30:00 PM	Monday	RC Dance Arts	23
2	2	Jazz	12:15:00 PM	2:15:00 PM	Tuesday	RC Dance Arts	25
3	3	Jazz II	2:15:00 PM	3:00:00 PM	Tuesday	RC Dance Arts	25
4	4	Ballet I	4:30:00 PM	5:30:00 PM	Monday	RC Dance Arts	25
5	5	Tap II	4:45:00 PM	5:30:00 PM	Tuesday	Rapid City	30
6	6	Me & My Tot	11:00:00 AM	11:30:00 AM	Tuesday	RC Dance Arts	15
7	7	Kinderdance	3:00:00 PM	4:00:00 PM	Tuesday	RC Dance Arts	15
8	8	Mc	9:00:00 AM	10:00:00 AM	Sunday	RC Dance Arts	12

Figure 8.8: Class Search Window

### 8.1.2.b Manage Classes

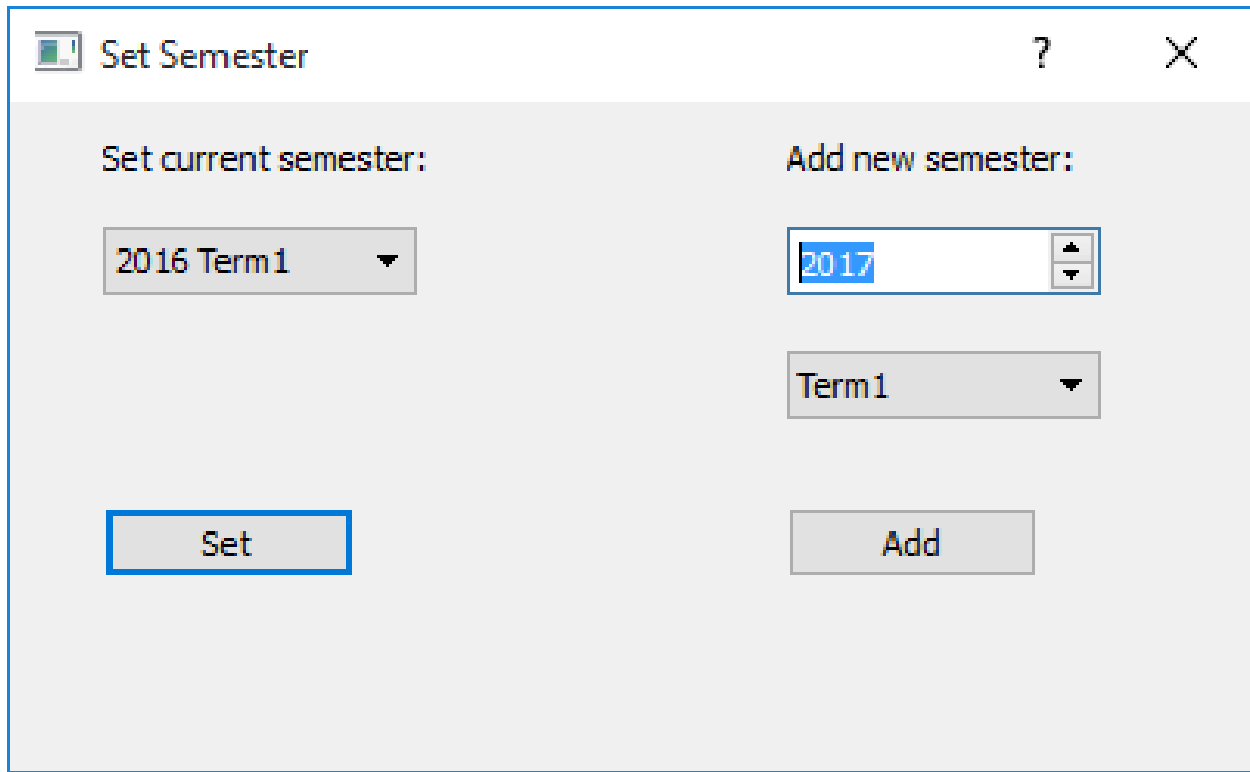
#### View Classes

This button allows the user to search classes like in the teacher search explained above. The general functionality is the same, the user is able to search by class name and refresh the view window after a search. The user can check the exact button if they wish to look only for what they entered. The Advanced search button allows the user to search for classes based on id, name location and time. Once the desired class is found the user can select a class and press the details button in order to view the full details for a course. Within this details page the user can modify the information and send updates to the system.

#### Add a Class:

A form will pop up where the user can enter in all the information needed to add a class. Once the user has entered in all the information and clicked submit a dialog will pop up where the user can check and confirm the submission. Selecting "Add New Location" in the location drop down will pop up a dialog box which will allow the user to enter in a new location for the class, the user can also select a location that already exist in the system.

#### Remove Class:



The screenshot shows a window titled "Set Semester". It has a title bar with a question mark icon and a close button (X). The window is divided into two main sections. The left section is titled "Set current semester:" and contains a dropdown menu with "2016 Term1" selected and a "Set" button below it. The right section is titled "Add new semester:" and contains a text input field with "2017" and a dropdown menu with "Term1" selected, with an "Add" button below it.

Figure 8.9: Set Semester Window

The user is able to select the name of a class and remove the selected class from the system. This removal is complete, this means that if the user removes a class it will completely remove all the data from the system connected to that class. This includes class data, removal from schedules, removal from any billing calculations, and any pending registrations

Set Semester:

Allows the user to select the current semester in the system or add a new semester.

### 8.1.2.c Manage Students

Search Students:

When the user selects search student, the system will bring up a window that displays the students in the system. The user can then type a student's name into the search bar and the field will display the students that names contain the entered value. If the user is looking for an exact match then they can check the "exact check box. After the user conducts a search, they can click the refresh button to bring back the full list of students in the database.

By clicking the advanced search button the user can further refine the search value, the advanced search dialog is accessed by clicking the advanced search button. Advanced search allows the user to to search for students by id, name, phone, guardian or date of birth.

One the users desired student is found the user can click on the name in the main window and click a details button to view all the information for that student. In the details window the user is able to change the information for an entry and submit the updates to the system.

Student Credits:

This button will pull up a window where the user can select a students name and their credit amount within

Search

Enter Student Name ☐ Exact

**Search** Advanced Search Refresh

	ID	Name	Gender	Date of Birth	Phone	Primary Guardian
1	1	Tom Berger	Male	11/1/2004	605-555-5556	None
2	2	Alice Corwin	Female	11/13/2003	605-555-5555	Tim
3	3	Petra Protava	Male	4/12/2005	605-555-5555	Jim
4	4	Kate Logar	Female	11/10/2006	510-366-1666	Amy
5	5	Tyler Cole	Male	2/5/2015	605-555-5557	Jim

Detail Back

Figure 8.10: Student Search Window

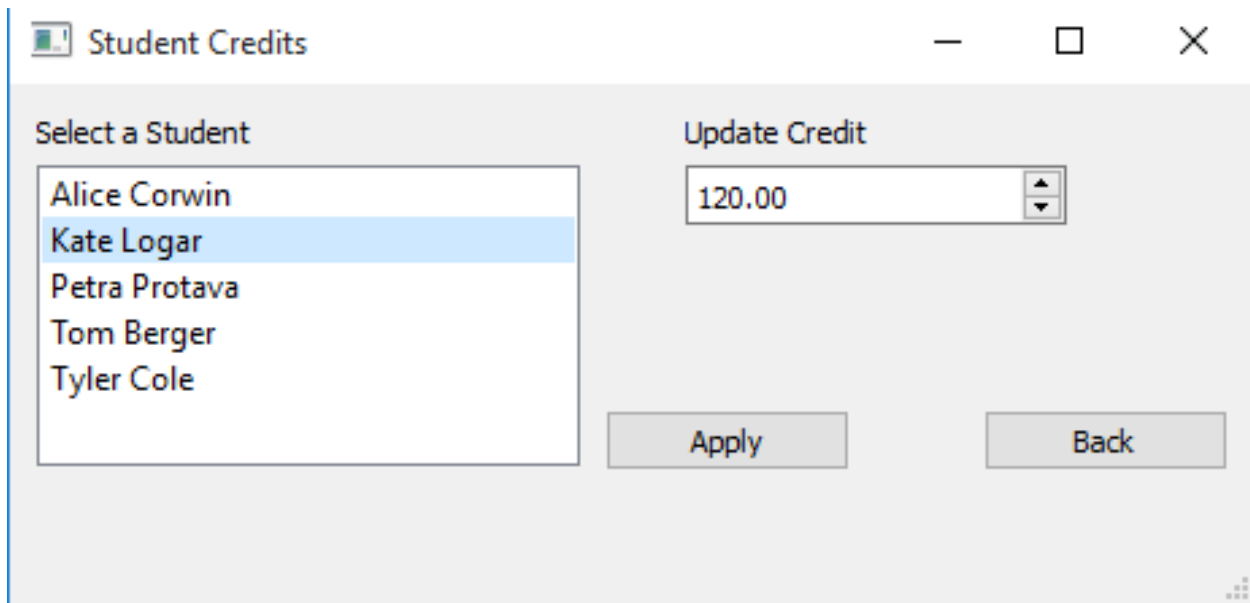


Figure 8.11: Student Credit Window

the system will be displayed. The user can then update the credit amount for that student. The credits are not factored into any student billing calculations. This is in accordance with the Academy of Dances Arts refund and credit policy as the credits and refund distribution is up to the users of the system. One the credits have been used the user must reset the reset the students credits to zero.

#### Add/Remove Student:

This button brings up a window that allows the user to select a class. Once a class is selected the students currently registered will appear along with their class approval status. User can then select a student name and either click the add button to set the approval to one meaning the student is approved to take the class. A user can also remove the student which sets the approval to negative one meaning rejected. An approval of zero means the students registration is pending.

This function was created for the student interface links, however this set of functions were dropped so this function was left in at the request of the project's initial client Dr. Jeff McGough, so a future iteration of the project could see it. However in the projects current state this window serves no real overall purpose and should not be used by any users of the software. Anyone wishing to do student registration should do so through the student registration function explained below.

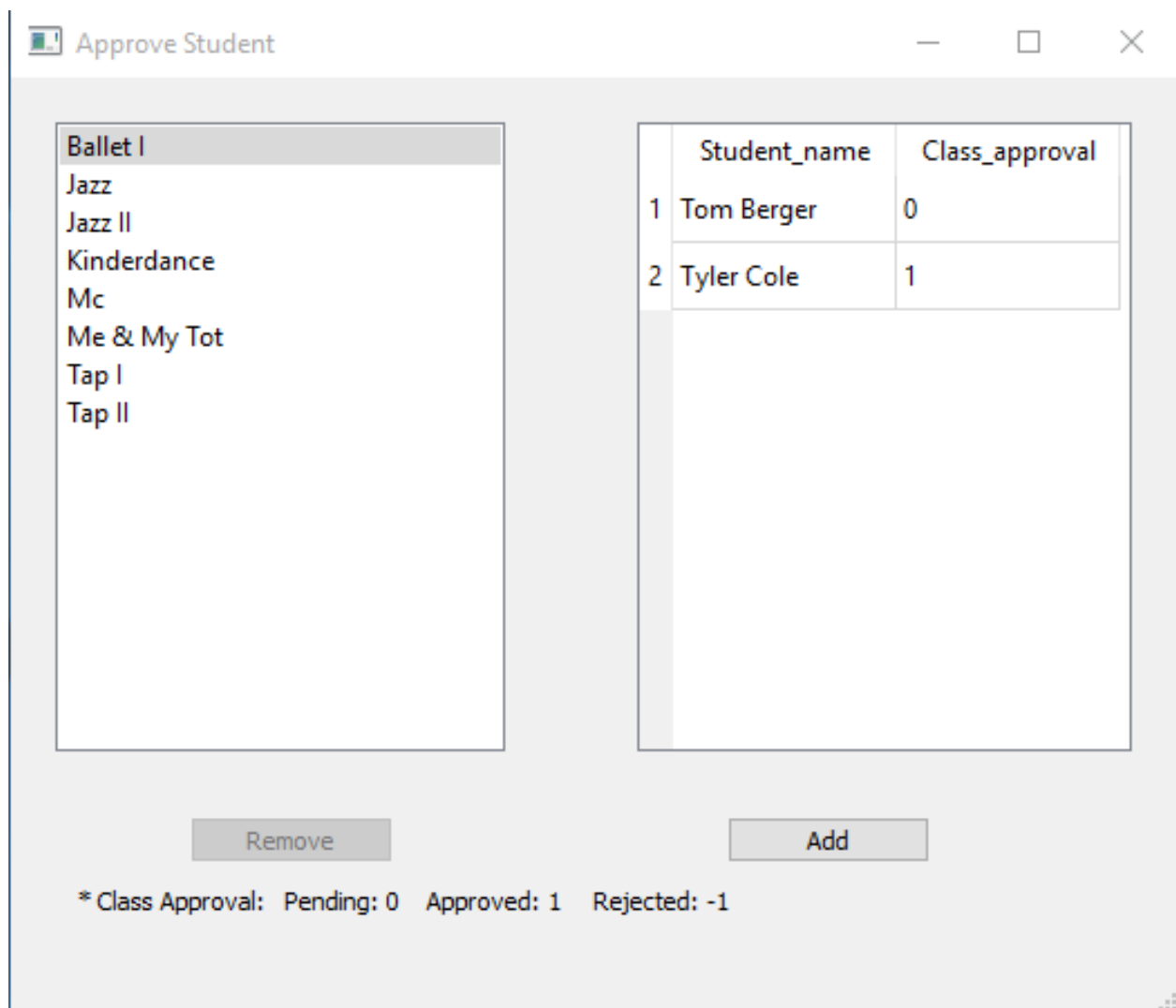
#### Registration:

The user is able search for a preexisting and select a student. The user can then click the "Update" button to bring up the details of that student, the user can then change and update the student's information in the system. Once the updates are made or if there are no information updates the user can click the "list" button and bring up a list of classes the student is currently registered for and the classes the student can register for. Classes can be dropped by selecting the class and clicking the drop button. Classes can be added from the pending class section using the add button.

Back on the student list window instead of clicking the "Update" button, the user can click the "Add" Button and enter in a new students information. If the new students guardian is not in the system the user can click "Add Primary" to add the student's primary guardian. If the student needs a new secondary guardian the user can click the "Add Secondary" button and add another guardian to the system. Once all the information for the student has been added the user can click the "Add" button to finalize the student.

#### Remove Student:

The user is able to select the name of a student using a search bar and remove the selected student from the system. This removal is complete, this means that if the user removes a student it will completely remove



The screenshot shows a window titled "Approve Student" with a list of classes on the left and a table of student approvals on the right. The classes listed are Ballet I, Jazz, Jazz II, Kinderdance, Mc, Me & My Tot, Tap I, and Tap II. The table has two columns: Student\_name and Class\_approval. The first row shows Tom Berger with a Class\_approval of 0. The second row shows Tyler Cole with a Class\_approval of 1. Below the table, there are buttons for "Remove" and "Add". At the bottom, a status bar shows: \* Class Approval: Pending: 0 Approved: 1 Rejected: -1.

	Student_name	Class_approval
1	Tom Berger	0
2	Tyler Cole	1

\* Class Approval: Pending: 0 Approved: 1 Rejected: -1

Figure 8.12: First Student Registration

The image shows a software window titled "Class Registration" with a standard Windows-style title bar (minimize, maximize, close buttons). The window is divided into two main sections, each containing a table with four columns: Teacher\_name, Class\_id, Class\_name, and Class\_time.

**Class registered**

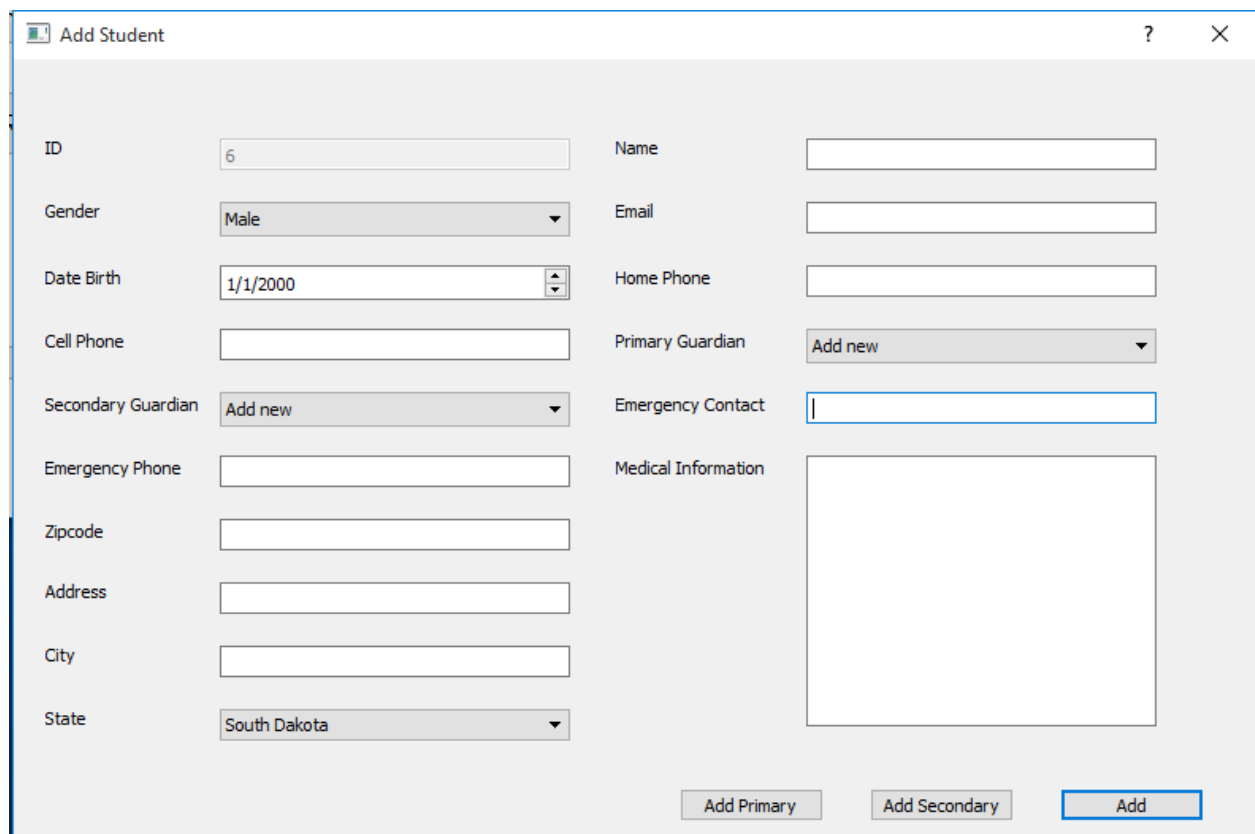
	Teacher_name	Class_id	Class_name	Class_time
1	Jamie Witmore	1	Tap I	3:30:00 PM
2	Scott Carda	2	Jazz	12:15:00 PM
3	Jamie Witmore	4	Ballet I	4:30:00 PM
4	Scott Carda	7	Kinderdance	3:00:00 PM
5	Jamie Witmore	8	Mc	9:00:00 AM

**Available classes**

	Teacher_name	Class_id	Class_name	Class_time
1	Scott Carda	3	Jazz II	2:15:00 PM
2	Scott Carda	5	Tap II	4:45:00 PM
3	Jamie Witmore	6	Me & My Tot	11:00:00 AM

At the bottom of the window, there are two buttons: "Drop" under the "Class registered" table and "Add" under the "Available classes" table.

Figure 8.13: Current Student Registration Window



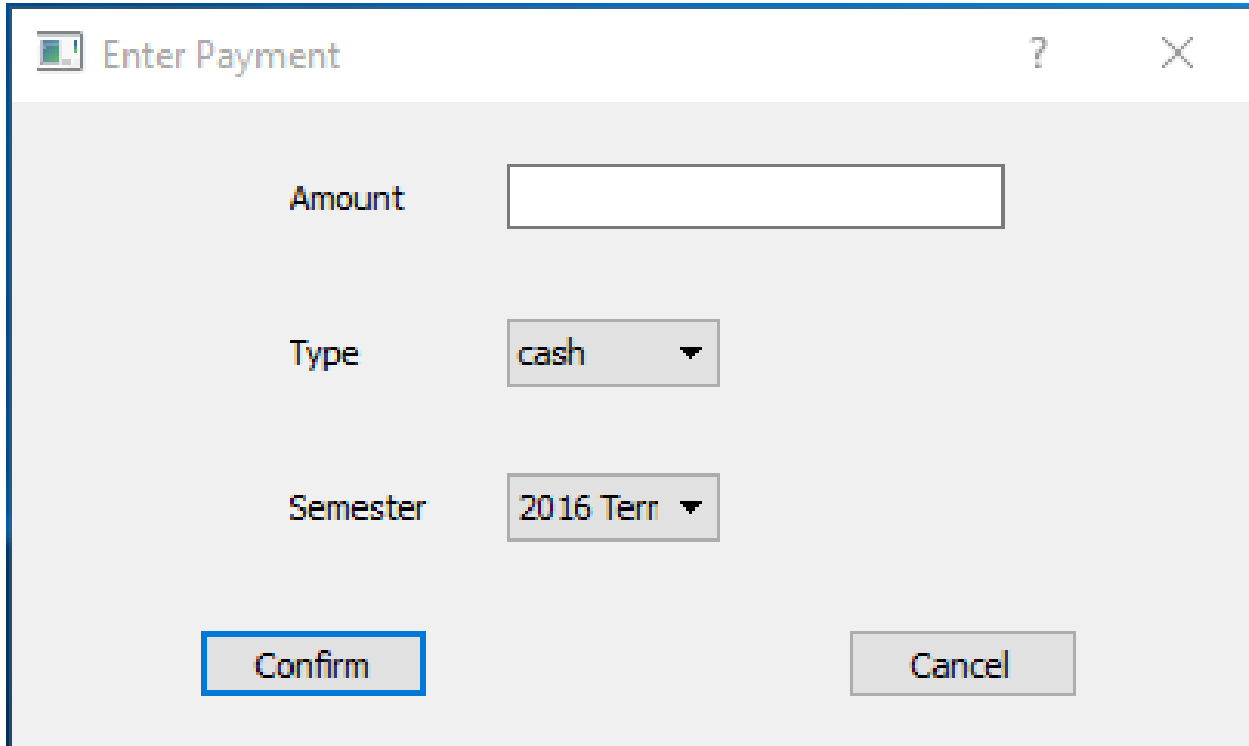
The image shows a software window titled "Add Student" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form for adding a new student. The form is organized into two columns of fields. The left column includes fields for ID (text input with "6"), Gender (dropdown menu with "Male"), Date Birth (calendar icon with "1/1/2000"), Cell Phone (text input), Secondary Guardian (dropdown menu with "Add new"), Emergency Phone (text input), Zipcode (text input), Address (text input), City (text input), and State (dropdown menu with "South Dakota"). The right column includes fields for Name (text input), Email (text input), Home Phone (text input), Primary Guardian (dropdown menu with "Add new"), Emergency Contact (text input), and Medical Information (a large text area). At the bottom right of the window, there are three buttons: "Add Primary", "Add Secondary", and "Add". The "Add" button is highlighted with a blue border.

ID	6	Name	
Gender	Male	Email	
Date Birth	1/1/2000	Home Phone	
Cell Phone		Primary Guardian	Add new
Secondary Guardian	Add new	Emergency Contact	
Emergency Phone		Medical Information	
Zipcode			
Address			
City			
State	South Dakota		

Buttons: Add Primary, Add Secondary, Add

Figure 8.14: Add New Student Window





The screenshot shows a window titled "Enter Payment". It has a standard Windows-style title bar with a question mark icon and a close button (X). The main area of the window is light gray and contains three labeled input fields arranged vertically. The first field is labeled "Amount" and is a simple text box. The second field is labeled "Type" and is a dropdown menu currently showing "cash". The third field is labeled "Semester" and is a dropdown menu currently showing "2016 Terr". At the bottom of the window, there are two buttons: "Confirm" on the left and "Cancel" on the right. The "Confirm" button is highlighted with a blue border.

Figure 8.15: Partial Payment Window

all the data from the system connected to that student. This includes class data, removal from schedules, removal from any billing calculations, any pending registrations, and if no other students such as sibling are connected to an address or a guardian the address and/or guardians will be removed as well. Any remove functions should be done with caution

#### 8.1.2.d Manage Billing

These functions are the general billing function for this version of the software and allow the user to handle the base level billing functionality of the DanceSoft project.

##### Enter Partial Payment:

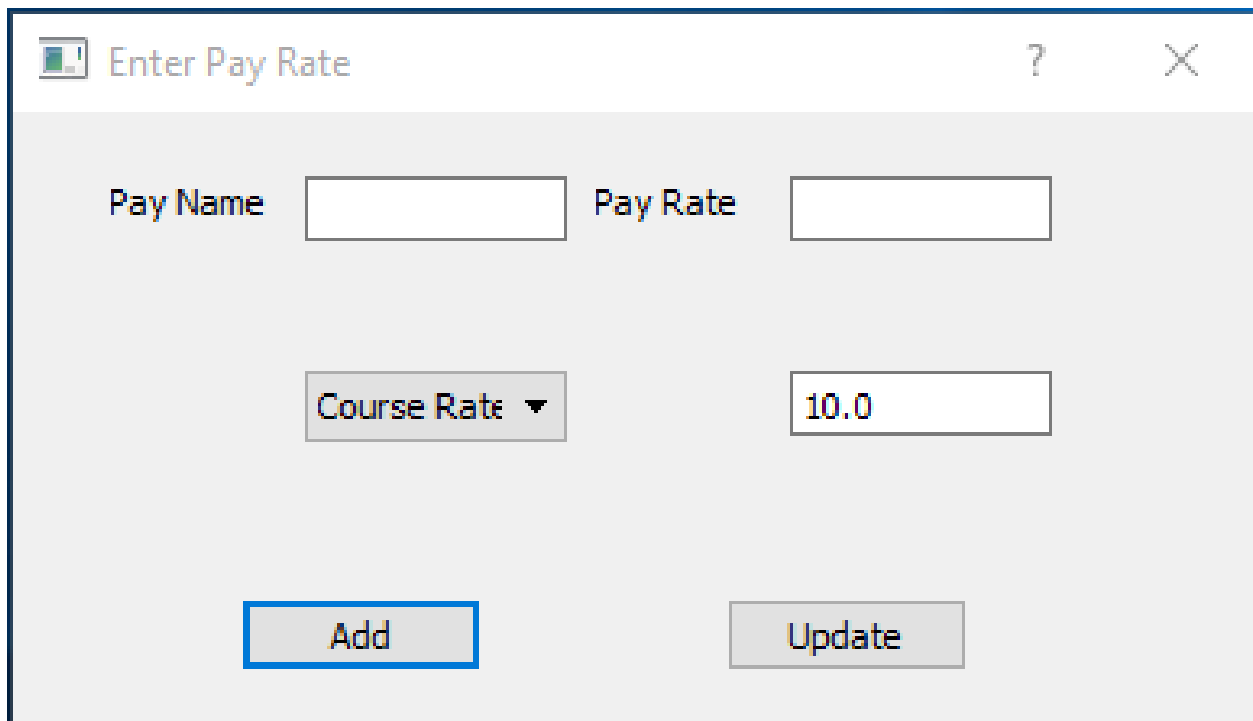
The User is able to search for a student and select them. When the user selects a student that is making a payment a window will appear where the user can type in the amount paid by the student. The type of payment is selected from a drop down menu, this value is used in the billing history to display how the student paid. The last combo box allows the user to select the term in which the payment was made. The payment is processed when the user clicks "Confirm."

##### Enter Full Payment:

Enter full payment works the same way as partial payment the user is able to search for and select a student, however instead of entering a payment by clicking the button the student's owed amount will be set to zero.

##### Enter Teacher Pay Rate:

The Enter Teacher Pay Rate button opens up a window where the user can select a teacher. Once a teacher is selected the user can open the pay rate window. From the pay rate window the user can enter a new pay rate for the current teacher and the amount that pay rate is worth. The user can then add the pay rate to the system by clicking the "Add" button. If the user chooses to update the pay rate the user can select the



The screenshot shows a window titled "Enter Pay Rate". Inside the window, there are two input fields at the top: "Pay Name" and "Pay Rate". Below these, there is a "Course Rate" dropdown menu and a text box containing the value "10.0". At the bottom of the window, there are two buttons: "Add" and "Update". The "Add" button is highlighted with a blue border.

Figure 8.16: Enter Pay Rate Window

rate they wish to update from the drop down menu and change the value in the box. The update is submitted by clicking the update button. Currently the added pay rates are for that teacher, but the course rate is the same among all teachers. Future versions should add the ability to have different course rates for each teacher.

#### Student Balance:

The user selects a student and can click the "Statement" button. The student's statement will produce a name, the current semester, the classes taken that semester, the total money paid, due, and the remaining balance. The user can print out the statement as well if needed.

#### Billing History:

When the user clicks the "Billing History" button, a window appears where the users can search for and select a student. Once a student is selected the user can click the statement button and produce the payment list for a student. The payment list contains the name of the student, the term the payment was made for, the amount paid, and the date the payment was made. The user can print the statement if they want to using the print button.

#### Tuition Rates and Fees:

The user is able to view all the tuition rates in the system and update them if need be, by changing the values of in the boxes. The user can also add and remove tuition rates as well. In the fee window the user can add a fee update a fee in the system and remove a fee if they so choose. However as of this version of the project none of the fees are factored in to any of the calculations. In future versions the project should be able to connect the fees to billing calculations. The tuition rates are based on the amount of minutes and they in turn determine the amount a student owes. If these are changed or remove it could mess with the way the system does billing for the Academy. Handle with caution. In future iterations teams should be able to better connect these systems.

### 8.1.3 Teacher Landing

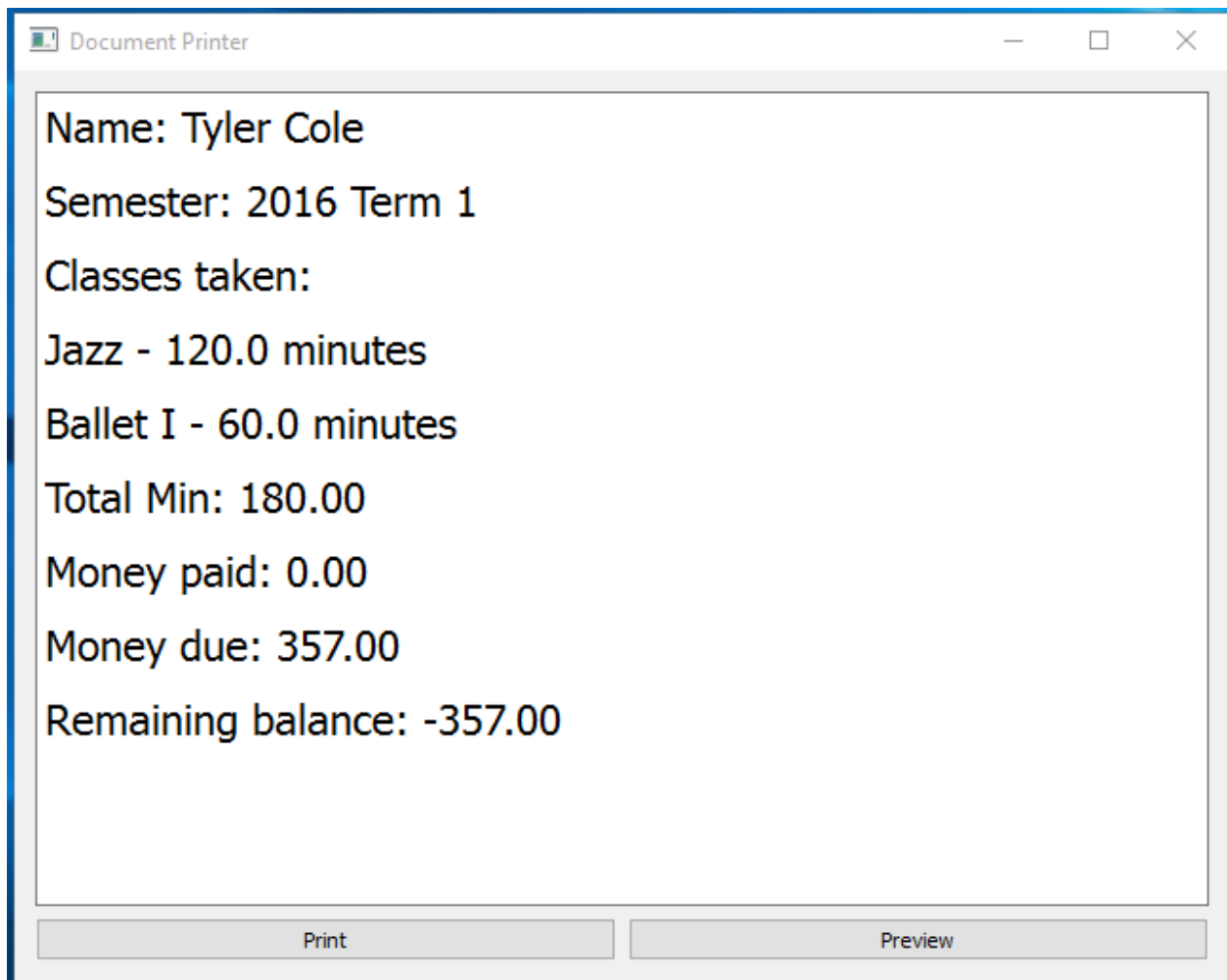


Figure 8.17: Student Balance Window

Document Printer

PaymentId	Name	Term	Amount paid	Date
83	Tom Berger	2016 Term 1	100	Mon Apr 18 2016
81	Tom Berger	2016 Term 1	-200	Fri Apr 8 2016
82	Tom Berger	2016 Term 2	100	Fri Apr 8 2016
80	Tom Berger	2016 Term 1	300	Fri Apr 8 2016

Print Preview

Figure 8.18: Student Billing History

Tuition Rates

Tuition Description

Tuition Cost: 0.00

Time: 0 ☐ Minutes ☐ Hours

Add Update Remove

Refresh List Cancel

Fee Rates

Fee/Discount Description

Cost: 0.00

Add Update Remove

Refresh List Cancel

4-Week Dance Sessions  
Additional Family Member  
Annual Discount Percent  
Certificate for 3 x 45-minute sessio  
Discounted Family Plan  
Early Discount Percent  
Interest Rate  
Late Fee

Figure 8.19: Tuition and Fees

	ID	Name	Gender	Date of Birth	Phone	Primary Guardian
1	1	Tom Berger	Male	11/1/2004	605-555-5556	None
2	2	Alice Corwin	Female	11/13/2003	605-555-5555	Tim
3	3	Petra Protava	Male	4/12/2005	605-555-5555	Jim
4	4	Kate Logar	Female	11/10/2006	510-366-1666	Amy
5	5	Tyler Cole	Male	2/5/2015	605-555-5557	Jim

Figure 8.20: Student Search Window

### 8.1.3.a Student Options

#### Search Students:

When the user selects search student, the system will bring up a window that displays the students in the system. The user can then type a student's name into the search bar and the field will display the students that names contain the entered value. If the user is looking for an exact match then they can check the "exact check box. After the user conducts a search, they can click the refresh button to bring back the full list of students in the database.

By clicking the advanced search button the user can further refine the search value, the advanced search dialog is accessed by clicking the advanced search button. Advanced search allows the user to to search for students by id, name, phone, guardian or date of birth.

One the users desired student is found the user can click on the name in the main window and click a details button to view all the information for that student. In the details window the user is able to change the information for an entry and submit the updates to the system.

#### Add to Class:

This button brings up a window that allows the user to select a class. Once a class is selected the students currently registered will appear along with their class approval status. User can then select a student name

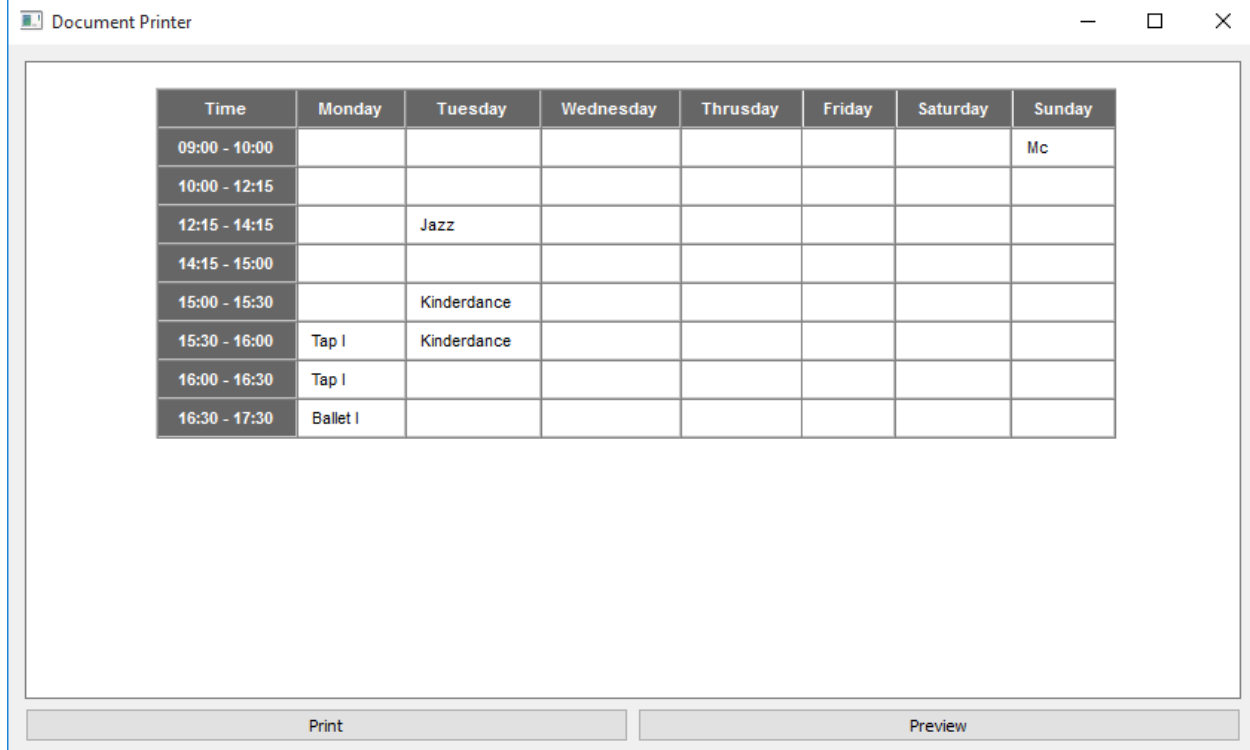


Figure 8.21: Student Schedule Window

and either click the add button to set the approval to one meaning the student is approved to take the class. A user can also remove the student which sets the approval to negative one meaning rejected. An approval of zero means the students registration is pending.

This function was created for the student interface links, however this set of functions were dropped so this function was left in at the request of the project's initial client Dr. Jeff McGough, so a future iteration of the project could see it. However in the projects current state this window serves no real overall purpose and should not be used by any users of the software. Anyone wishing to do student registration should do so through the student registration function explained below.

#### Student Schedule:

This button allows the user to pull up a window where they can select a student and view there class schedule. The user can also print out their schedule by pressing the print button. The schedule is design to only display the times that the student is in a class.

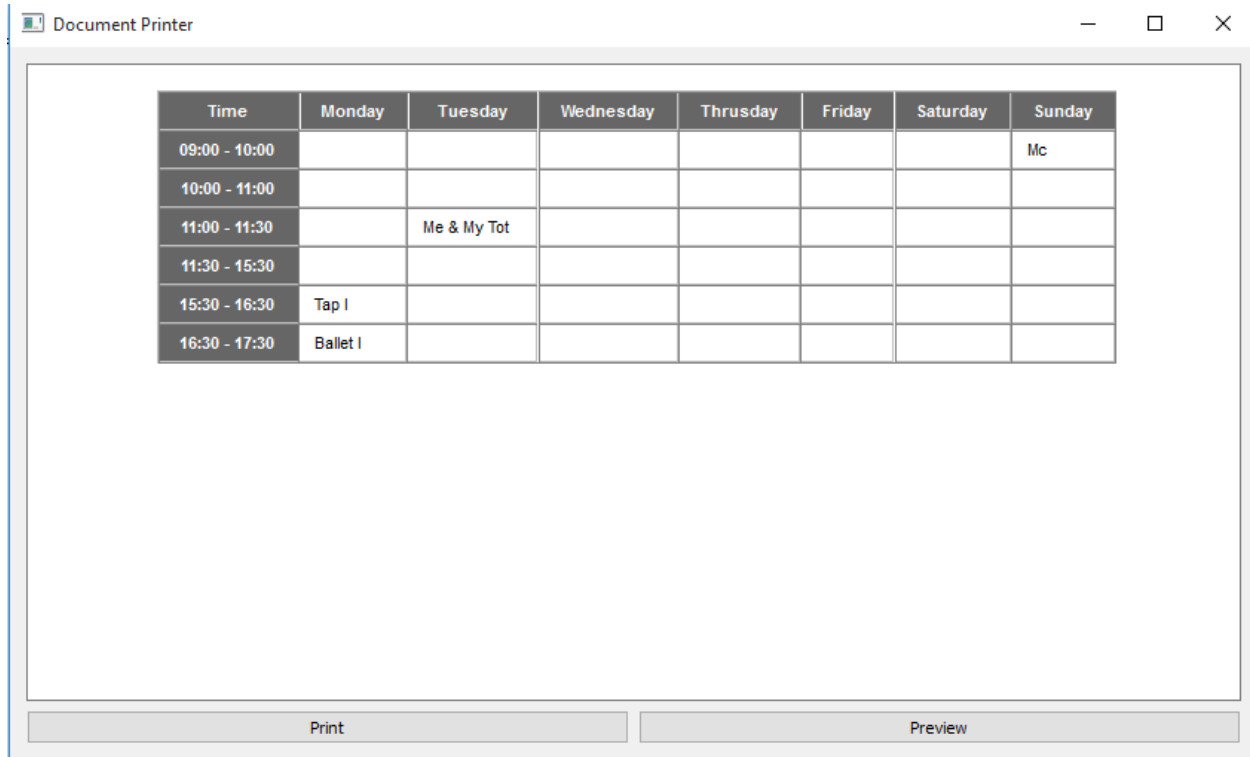
### 8.1.3.b Class Options

This section contains the class related function that a teacher may need to manage their day to day activities.

#### View Classes:

This button allows the user to search classes like in the teacher search explained in the admin section. The general functionality is the same, the user is able to search by class name and refresh the view window after a search. The user can check the exact button if they wish to look only for what they entered. The Advanced search button allows the user to search for classes based on id, name location and time. Once the desired class is found the user can select a class and press the details button in order to view the full details for a course. Within this details page the user can modify the information and send updates to the system.

#### Class Schedule:



Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
09:00 - 10:00							Mc
10:00 - 11:00							
11:00 - 11:30		Me & My Tot					
11:30 - 15:30							
15:30 - 16:30	Tap I						
16:30 - 17:30	Ballet I						

Figure 8.22: Teacher Schedule

This button allows the user to pull up a window where they can select a teacher and view their teaching schedule. The user can also print out their schedule by pressing the print button. The schedule is designed to only display the times that the teacher is teaching a class.

#### Class Role Sheet:

This button creates a window which contains the classes the logged in teacher is currently teaching. The user can click on a class to view the students in the class. If the user wants a formatted role sheet they can click the print button which allows the user to print out a role sheet for the class. The role sheet contains the student's name, and emergency contact information, along with a check box for each week in a three-month period. This format is in accordance with the one a week classes offer at the Academy of Dance Arts.

### 8.1.3.c Personal Options

This section contains the functions connected to modify information directly connected to the user. These functions include change password, change user name, modify personal information, and enter hours.

#### Change Password

When using this function the user is prompted with a dialog box, from this dialog the user will enter their user name, their desired new password, and confirm the new password. The user will be prompted to reenter their password if their user name or password are incorrect.

#### Change Username:

When using this function the user is prompted with a dialog box, from this dialog the user will enter their user name, their password, and the desired new user name. The user will be prompted to reenter their user name if their user name or password are incorrect.

#### Modify Personal Information:

Document Printer

### Attendance Sheet

ID	Name	Home Phone	Emergency Contact	Emergency Phone	Month 1	Month 2	Month 3
4	Kate Logar	510-366-1666	Evet	555-555-5555	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	Petra Protava	605-555-5555	Nicole	555-555-5555	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	Tom Berger	605-555-5556	Jerry	555-555-5555	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Print Preview

Figure 8.23: Class Role Sheet



## Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
<a href="#">Gzipped source tarball</a>	Source release		e80a0c1c71763ff6b5a81f8cc9bb3d50	19435166	<a href="#">SIG</a>
<a href="#">XZ compressed source tarball</a>	Source release		8d526b7128affed5f5be72ceac8d2fc63	14307620	<a href="#">SIG</a>
<a href="#">Mac OS X 32-bit i386/PPC installer</a>	Mac OS X	for Mac OS X 10.5 and later	8491d013826252228ffcdeda0d9348d6	24829047	<a href="#">SIG</a>
<a href="#">Mac OS X 64-bit/32-bit installer</a>	Mac OS X	for Mac OS X 10.6 and later	349c61e374f6aeb44ca85481ee14d2f5	23170139	<a href="#">SIG</a>
<a href="#">Windows debug information files</a>	Windows		d6ffcb8cdabd93ed7f2feff661816511	37743788	<a href="#">SIG</a>
<a href="#">Windows debug information files for 64-bit binaries</a>	Windows		a0eea5b3742954c1ed02bddf30d07101	25038530	<a href="#">SIG</a>
<a href="#">Windows help file</a>	Windows		5fa4e75dd4edc25e33e56f3c7486cd15	7461732	<a href="#">SIG</a>
<a href="#">Windows x86-64 MSI installer</a>	Windows	for AMD64/EM64T/x64, not Itanium processors	963f67116935447fad73e09cc561c713	26054656	<a href="#">SIG</a>
<a href="#">Windows x86 MSI installer</a>	Windows		e96268f7042d2a3d14f7e23b2535738b	24932352	<a href="#">SIG</a>

Figure 8.24: Python zip files

If the user ever needs to change their personal information they can click this button to bring up a form containing their personal information. In this form the user can change their name, date of birth, phone numbers, address, and other information within the system. Like the other update teacher form in the admin section of the DanceSoft project, if the user leaves any of the required fields open they will be prompted to fill them in. Also if the user changes their address they will be given a dialog box asking if they would like to update their existing address or create a new one within the database. This is done because if two people going to, or working at the Academy live at the same address the teacher can create a new entry so that the other person address does not get effected. However if the teacher is the only one living at the address the system does not want to create dead data within the database, so the user is given an ability to change the entry.

As stated the user should use caution when ever changing address because if a user updates an address that someone else lives at then that person will have incorrect data, also if a teacher is the only person in the database with an address and a new address is added instead of updating then dead data will be created, and in this project iteration a user would need to access the database directly to remove a dead address as address removal is linked to teacher and student removal.

Enter Hours:

The user is able to look through the pay rates linked to their account by system admin and log hours for the various pay rates.

## 8.2 Installation Guide

The following is the steps to install the DanceSoft software on a users machine:

**Step 1:** The first thing a user needs to do is make sure that a valid version of Python 3 can run on their system, if you already have python 3 installed on your machine please skip to step 2. The most recent version of Python 3 can be found on <https://www.python.org/downloads/> the user can then click on the download link and download a python zip file that is compatible with the operating system being used by the user. Follow the instruction on the install, once the install is complete the python files should be located in your local drive unless the user specified a different directory during install. A user has several ways of confirming that python installed correctly on their system. If the user is with the command prompt they can run the Python 3 command to confirm successful installation. If the user would like to confirm using non-command line, the user can go to the python 3 file on their drive and run the Python.exe file. If a command window appears then the user has successfully installed python and can proceed to step 2.

**Step 2:** The next step is to install the PyQt libraries and the designer so the user can run the scripts containing PyQt code. There are several versions of PyQt, the version the DanceSoft team used was PyQt4. PyQt4 can be downloaded from <https://www.riverbankcomputing.com/software/pyqt/download>, once there the user can select the zip file that goes with their operating system. The website also provides

stable windows installers if the user is running a windows system. If the installer is used the libraries will automatically be stored in the site-packages sub-directory of the python folder and the user should be able to start using PyQt libraries. If the user is not running windows, they can download and use the brew facility to easily install the need files. If the user runs **brew install PyQt --with-python3** from the OS X command line the system should automatically install the needed files. Otherwise the user will need to download the snippets from <https://www.riverbankcomputing.com/software/pyqt/download> and install the PyQt libraries in the site-packages folder.

**Step 3:** The last major install a user will need to do before using the software is the MySQL database software necessary to use the SQL components of the DanceSoft project. MySQL can be installed by following the download instructions on <https://www.mysql.com/downloads/> and downloading the free version of the software. This installer will install all the tools needed to manipulate and use the database directly if necessary.

The user should now have all the needed tools installed to run the DanceSoft Software.

9

---

## Class Index

---



# 10

---

## Class Documentation

---

### 10.1 Add Class Reference

#### Public Member Functions

- `def __init__ (self):`
- `def setup_database (self):`
- `def add_location (self):`
- `def add_class (self):`
- `def conn (self):`

#### 10.1.1 Constructor & Destructor Documentation

##### 10.1.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

#### 10.1.2 Member Function Documentation

##### 10.1.2.a `def setup_database (self):`

Set up a connection with database. :

- `Add_a_Class.py`

##### 10.1.2.b `def add_location (self):`

Add a new location to current window but not add to database. :

- `Add_a_Class.py`

##### 10.1.2.c `def add_class (self):`

Read name, cost, start time, end time, start date, end date, location, capacity, clothing and description of class from various fields and then it will be written to database while doing error checks. :

- `Add_a_Class.py`

##### 10.1.2.d `def conn (self):`

Set up a connection with database. :

- `Add_a_Class.py`

## 10.2 Add admin Reference

### Public Member Functions

- `def __init__ (self):`
- `def submit (self):`

#### 10.2.1 Constructor & Destructor Documentation

##### 10.2.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database.

#### 10.2.2 Member Function Documentation

##### 10.2.2.a `def submit (self):`

submit the admin information to database of user entered and . :

- `addAdmin.py`

## 10.3 Add new teacher Reference

### Public Member Functions

- `def __init__ (self):`
- `def check__existing__address (self):`
- `def insert__teacher (self):`
- `def create__account (self):`
- `def conn (self):`

#### 10.3.1 Constructor & Destructor Documentation

##### 10.3.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database.

#### 10.3.2 Member Function Documentation

##### 10.3.2.a `def check__existing__address (self):`

Check weather it has existed address in the database if it has let user choose weather they want to add new address or update existing address. :

- `Add_new_teacher.py`

##### 10.3.2.b `def insert__teacher (self):`

It reads data of teacher's profile from window while doing error checking by utilizing regular expression and submit to the database if the info is correctly filled. :

- `Add_new_teacher.py`

##### 10.3.2.c `def create__account (self):`

It will create an account is associated with current inserted teacher and then add account with appropriate permission level. :

- `Add_new_teacher.py`

**10.3.2.d def conn (self):**

Set up a database connection at programming started. :

- Add\_new\_teacher.py

**10.4 addLocation Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def addLocation (self):
- def getLocation (self):
- def setClose (self):
- def getClose (self):

**10.4.1 Constructor & Destructor Documentation****10.4.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database.

**10.4.2 Member Function Documentation****10.4.2.a def addLocation (self):**

Add new location to current GUI page, which reads from user entered, and then transfer it to upper case. :

- addLocation.py

**10.4.2.b def getLocation (self):**

In order to let main window can get the result location.

:

- addLocation.py

**10.4.2.c def setClose (self):**

Set flag for later usage. :

- addLocation.py

**10.4.2.d def getClose (self):**

Get flag from previous setted flag. :

- addLocation.py

**10.5 Addstu details Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def set\_student (self):
- def add\_student (self):

### 10.5.1 Constructor & Destructor Documentation

#### 10.5.1.a def `__init__` (self):

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.5.2 Member Function Documentation

#### 10.5.2.a def `set__student` (self):

Set class to be approved

- Addstu\_detail\_dialog.py

#### 10.5.2.b def `add__student` (self):

Get user clicked student's name. :

- Addstu\_detail\_dialog.py

## 10.6 Addstu window Reference

### Public Member Functions

- def `__init__` (self):
- def `add__student` (self):
- def `remove__student` (self):
- def `select__student` (self):
- def `show__student` (self):
- def `conn` (self):

### 10.6.1 Constructor & Destructor Documentation

#### 10.6.1.a def `__init__` (self):

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.6.2 Member Function Documentation

#### 10.6.2.a def `add__student` (self):

Get current clicked student name and then open a new window for further operation.

- Addstu\_window.py

#### 10.6.2.b def `remove__student` (self):

Drop selected classes of student and the calculate the amount refund. :

- Addstu\_window.py

#### 10.6.2.c def `select__student` (self):

Choose selected students and show the approval status. :

- Addstu\_window.py



**10.6.2.d def show\_\_student (self):**

Populate all the status to the Table View. :

- Addstu\_window.py

**10.6.2.e def conn (self):**

Set up a database connection. :

- Addstu\_window.py

## 10.7 Addstu window admin Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def add\_\_student (self):
- def remove\_\_student (self):
- def select\_\_student (self):
- def show\_\_student (self):
- def conn (self):

### 10.7.1 Constructor & Destructor Documentation

**10.7.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.7.2 Member Function Documentation

**10.7.2.a def add\_\_student (self):**

Get current clicked student name and then open a new window for further operation.

- Addstu\_window\_admin.py

**10.7.2.b def remove\_\_student (self):**

Drop selected classes of student and the calculate the amount refund. :

- Addstu\_window\_admin.py

**10.7.2.c def select\_\_student (self):**

Choose selected students and show the approval status. :

- Addstu\_window\_admin.py

**10.7.2.d def show\_\_student (self):**

Populate all the status to the Table View. :

- Addstu\_window\_admin.py

**10.7.2.e def conn (self):**

Set up a database connection. :

- Addstu\_window\_admin.py

## 10.8 Admin Reference

### Public Member Functions

- `def __init__ (self):`
- `def logout (self):`
- `def registration (self):`
- `def enter_teacher_payrate (self):`
- `def enter_teacher_hour (self):`
- `def show_admin_list (self):`
- `def student_balance (self):`
- `def enter_full_payment (self):`
- `def enter_partial_payment (self):`
- `def set_semester (self):`
- `def teaching_his (self):`
- `def billing (self):`
- `def fee (self):`
- `def tuition (self):`
- `def add_teacher (self):`
- `def removeTeach (self):`
- `def assign_teacher (self):`
- `def update_teacher (self):`
- `def search_teacher (self):`
- `def view_class (self):`
- `def new_class (self):`
- `def search_student (self):`
- `def add_student (self):`
- `def register (self):`
- `def removeStu (self):`
- `def removeClass (self):`
- `def studentCredit (self):`
- `def change_window (self):`
- `def conn (self):`
- `def Back_btn (self):`
- `def Quit (self):`

### 10.8.1 Constructor & Destructor Documentation

#### 10.8.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.8.2 Member Function Documentation

#### 10.8.2.a `def logout (self):`

Log out from current system and then asks them to log in to system.

- Admin.py

#### 10.8.2.b `def registration (self):`

It allows user to either new student or using existed student for registering classes. :

- Admin.py

**10.8.2.c def enter\_teacher\_payrate (self):**

It allows admin to enter specific pay rate of different categories. or update existed rate. :

- Admin.py

**10.8.2.d def enter\_teacher\_hour (self):**

Enter how many hours they spend on each category. :

- Admin.py

**10.8.2.e def show\_admin\_list (self):**

It allows user to get a new window with a list of admin populated in Table View. :

- Admin.py

**10.8.2.f def student\_balance (self):**

It allows user to check balance of each student with classes taken and user also can print it out. :

- Admin.py

**10.8.2.g def enter\_full\_payment (self):**

It will pop up a window with a list of students who has not clear their payment and user can choose whether to clear rest of money they should pay for one or multiple student. :

- Admin.py

**10.8.2.h def enter\_partial\_payment (self):**

It allows user to enter a single payment for one or multiple students with different payment method. :

- Admin.py

**10.8.2.i def set\_semester (self):**

Set current semester for the system or add new semester to the system. :

- Admin.py

**10.8.2.j def teaching\_his (self):**

Display teacher's history of classes they taught at previous semesters. :

- Admin.py

**10.8.2.k def billing (self):**

Print out the billing history of student payment ordered by date in descending order. :

- Admin.py

**10.8.2.l def fee (self):**

User can add a new fee rate or update existed fee rate. :

- Admin.py

**10.8.2.m def tuition (self):**

User can add a new tuition rate for daily, weekly, monthly or update existed tuition rate. :

- Admin.py

**10.8.2.n def add\_\_teacher (self):**

It allows user to enter a new teacher to database. :

- Admin.py

**10.8.2.o def removeTeach (self):**

It allows user to remove teacher from database and other information related to selected teacher. :

- Admin.py

**10.8.2.p def assign\_\_teacher (self):**

It allows user to assign classes to teacher based on the time availability or change instructor for a class which is already assigned a teacher. :

- Admin.py

**10.8.2.q def update\_\_teacher (self):**

It allows teacher updating the information they have. :

- Admin.py

**10.8.2.r def search\_\_teacher (self):**

It allows user to search teacher information and look into their specific information or doing advance search based on different fields they have. :

- Admin.py

**10.8.2.s def view\_\_class (self):**

It allows user to search class information and look into their specific information or doing advance search based on different fields they have. :

- Admin.py

**10.8.2.t def new\_\_class (self):**

It allows user to add new class with its related information as well. :

- Admin.py

**10.8.2.u def search\_\_student (self):**

It allows user to search student information and look into their specific information or doing advance search based on different fields they have. :

- Admin.py

**10.8.2.v def add\_student (self):**

It allows user to add new student with its related information as well. :

- Admin.py

**10.8.2.w def register (self):**

It allows user to get approved by the instructor. :

- Admin.py

**10.8.2.x def removeStu (self):**

It allows user to remove student from database and other information related to selected student. :

- Admin.py

**10.8.2.y def removeClass (self):**

It allows user to remove class from database and other information related to selected class. :

- Admin.py

**10.8.2.z def studentCredit (self):**

It allows user to apply credit on student. :

- Admin.py

**10.8.3 Member Function continued Documentation****10.8.3.a def change\_window (self):**

It allows user to apply credit on student. :

- Admin.py

**10.8.3.b def conn (self):**

It allows user to create database connection. :

- Admin.py

**10.8.3.c def Back\_btn (self):**

Back to previous pages. :

- Admin.py

**10.8.3.d def Quit (self):**

Exit program. :

- Admin.py

**10.9 Admin info Reference****Public Member Functions**

- def \_\_init\_\_ (self):

### 10.9.1 Constructor & Destructor Documentation

#### 10.9.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. Populate the admin information in window.

## 10.10 Admin list Reference

### Public Member Functions

- `def __init__ (self):`
- `def add__Admin (self):`
- `def remove__Admin (self):`
- `def search__Admin (self):`
- `def print__Detail (self):`
- `def select__Admin (self):`
- `def conn (self):`

### 10.10.1 Constructor & Destructor Documentation

#### 10.10.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.10.2 Member Function Documentation

#### 10.10.2.a `def add__Admin (self):`

Add new admin to database. :

- `Admin_list_window.py`

#### 10.10.2.b `def remove__Admin (self):`

Remove admin from database. :

- `Admin_list_window.py`

#### 10.10.2.c `def search__Admin (self):`

Search admin by their name and display result in table view. :

- `Admin_list_window.py`

#### 10.10.2.d `def print__Detail (self):`

Display specific information of admin in a new dialog. :

- `Admin_list_window.py`

#### 10.10.2.e `def select__Admin (self):`

Get username when it has been clicked. :

- `Admin_list_window.py`

### 10.10.2.f def conn (self):

Set up a connection with database. :

- Admin\_list\_window.py

## 10.11 Adv class Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def checkbox\_change (self):

### 10.11.1 Constructor & Destructor Documentation

#### 10.11.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.11.2 Member Function Documentation

#### 10.11.2.a def checkbox\_change (self):

Detect weather check box has been checked in order to disable the text field. :

- Advsearch\_class\_Dialog.py

## 10.12 Adv student Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def checkbox\_change (self):

### 10.12.1 Constructor & Destructor Documentation

#### 10.12.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.12.2 Member Function Documentation

#### 10.12.2.a def checkbox\_change (self):

Detect weather check box has been checked in order to disable the text field. :

- Advsearch\_Dialog.py

## 10.13 Adv teacher Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def checkbox\_change (self):

### 10.13.1 Constructor & Destructor Documentation

#### 10.13.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.13.2 Member Function Documentation

#### 10.13.2.a `def checkbox_change (self):`

Detect weather check box has been checked in order to disable the text field. :

- Advsearch\_teacher.py

## 10.14 Assign class Reference

### Public Member Functions

- `def __init__ (self):`
- `def teacher_assignment (self):`
- `def assign_teacher (self):`
- `def Update_assign_teacher (self):`
- `def conn (self):`

### 10.14.1 Constructor & Destructor Documentation

#### 10.14.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.14.2 Member Function Documentation

#### 10.14.2.a `def teacher_assignment (self):`

Get selected teacher name and check the availability of time and then do action based on availability. :

- Assign\_class.py

#### 10.14.2.b `def assign_teacher (self):`

Assign teacher with selected class. :

- Assign\_class.py

#### 10.14.2.c `def Update_assign_teacher (self):`

Add new teacher to the class which already assigned a teacher. :

- Assign\_class.py



## 10.15 Billing history Reference

### Public Member Functions

- `def __init__ (self):`
- `def search_student (self):`
- `def print_history (self):`
- `def select_student (self):`
- `def conn (self):`

#### 10.15.1 Constructor & Destructor Documentation

##### 10.15.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

#### 10.15.2 Member Function Documentation

##### 10.15.2.a `def search_student (self):`

Search student by their and display result in the table view. :

- `billing_history_window.py`

##### 10.15.2.b `def print_history (self):`

Print student billing history in a new dialog ordered by date of they have paid. :

- `billing_history_window.py`

##### 10.15.2.c `def select_student (self):`

Get student name by user clicked. :

- `billing_history_window.py`

##### 10.15.2.d `def conn (self):`

Set up a connection with database. :

- `billing_history_window.py`

## 10.16 Print Hist Reference

### Public Member Functions

- `def __init__ (self):`
- `def handlePrint (self):`
- `def handlePreview (self):`
- `def handleTextChanged (self):`

#### 10.16.1 Constructor & Destructor Documentation

##### 10.16.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, populate html information onto the window.

### 10.16.2 Member Function Documentation

#### 10.16.2.a def handlePrint (self):

Connect to the printer and provide option for user printing :

- billing\_print.py

#### 10.16.2.b def handlePreview (self):

Generate pdf preview for user with other function. :

- billing\_print.py

#### 10.16.2.c def handleTextChanged (self):

enable button when text has been changed. :

- billing\_print.py

## 10.17 Change PW Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def update (self):
- def conn (self):

### 10.17.1 Constructor & Destructor Documentation

#### 10.17.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.17.2 Member Function Documentation

#### 10.17.2.a def update (self):

Update password of selected account. :

- change\_password\_window.py

#### 10.17.2.b def conn (self):

Set up a connection with database. :

- change\_password\_window.py

## 10.18 Change user Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def update (self):
- def getName (self):
- def conn (self):

### 10.18.1 Constructor & Destructor Documentation

#### 10.18.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.18.2 Member Function Documentation

#### 10.18.2.a `def update (self):`

Update username of selected account. :

- `change_username_window.py`

#### 10.18.2.b `def getName (self):`

Get username of current account. :

- `change_username_window.py`

#### 10.18.2.c `def conn (self):`

Set up a connection with database. :

- `change_username_window.py`

## 10.19 Class list Reference

### Public Member Functions

- `def __init__ (self):`
- `def add (self):`
- `def drop (self):`
- `def refresh (self):`

### 10.19.1 Constructor & Destructor Documentation

#### 10.19.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, populate classes in two table views with available classes and registered classes.

### 10.19.2 Member Function Documentation

#### 10.19.2.a `def add (self):`

Add one or more classes for selected student. :

- `Class_list_dialog.py`

#### 10.19.2.b `def drop (self):`

drop one or more classes for selected student. :

- `Class_list_dialog.py`

### 10.19.2.c def refresh (self):

refresh main window the list of student in order to get new student :

- Class\_list\_dialog.py

## 10.20 Class reg Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def update\_satus (self):
- def show\_student (self):
- def select\_student (self):
- def conn (self):

### 10.20.1 Constructor & Destructor Documentation

#### 10.20.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.20.2 Member Function Documentation

#### 10.20.2.a def update\_satus (self):

Choose weather approve or reject a class registered by the student. :

- class\_reg\_dialog.py

#### 10.20.2.b def show\_student (self):

Make the status to the database and it also enable the buttons when user choose a specific student. :

- class\_reg\_dialog.py

#### 10.20.2.c def select\_student (self):

Get student name, id and row number when user clicked a specific student :

- class\_reg\_dialog.py

#### 10.20.2.d def conn (self):

Set up a connection with database. :

- class\_reg\_dialog.py

## 10.21 Credits Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def applyCredit (self):
- def showStudentInformation (self):
- def conn (self):

### 10.21.1 Constructor & Destructor Documentation

#### 10.21.1.a def `__init__` (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.21.2 Member Function Documentation

#### 10.21.2.a def `applyCredit` (self):

It can either add a new credit to a student or update current student credit record. :

- `Credits_Window.py`

#### 10.21.2.b def `showStudentInformation` (self):

Populate a list of students and user can choose to see the details of each student by clicking on table view. :

- `Credits_Window.py`

#### 10.21.2.c def `conn` (self):

Set up a connection with database. :

- `Credits_Window.py`

## 10.22 Full pay Reference

### Public Member Functions

- def `__init__` (self):
- def `get_rate` (self):
- def `search_student` (self):
- def `clear_money` (self):
- def `select_student` (self):
- def `conn` (self):

### 10.22.1 Constructor & Destructor Documentation

#### 10.22.1.a def `__init__` (self):

In the constructor, it is established an UI interface, and populate student who has not been pay clear their payment onto the table view.

### 10.22.2 Member Function Documentation

#### 10.22.2.a def `get_rate` (self):

Get rate from each category. :

- `enter_fullpayment_window.py`

#### 10.22.2.b def `search_student` (self):

Search student by their name and populate to the table view. :

- `enter_fullpayment_window.py`

**10.22.2.c def clear\_\_money (self):**

Clear all the owe for one or more students. :

- enter\_\_fullpayment\_\_window.py

**10.22.2.d def select\_\_student (self):**

Enable the button. :

- enter\_\_fullpayment\_\_window.py

**10.22.2.e def conn (self):**

Set up a connection with database. :

- enter\_\_fullpayment\_\_window.py

## 10.23 Enter hours Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def show\_\_hours (self):
- def select\_\_teacher (self):
- def search\_\_teacher (self):
- def conn (self):

### 10.23.1 Constructor & Destructor Documentation

**10.23.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, populate students onto the table view.

### 10.23.2 Member Function Documentation

**10.23.2.a def show\_\_hours (self):**

Show hours of each category for selected student and populate in the text field. :

- enter\_\_hours\_\_window.py

**10.23.2.b def select\_\_teacher (self):**

Get the teacher name and enable the button when it has been clicked. :

- enter\_\_hours\_\_window.py

**10.23.2.c def search\_\_teacher (self):**

Search teacher by their name and then populate onto table view. :

- enter\_\_hours\_\_window.py

**10.23.2.d def conn (self):**

Set up a connection with database. :

- enter\_\_hours\_\_window.py

## 10.24 Print fees Reference

### Public Member Functions

- `def __init__ (self):`
- `def handlePrint (self):`
- `def handlePreview (self):`
- `def handleTextChanged (self):`

#### 10.24.1 Constructor & Destructor Documentation

##### 10.24.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, populate html information onto the window.

#### 10.24.2 Member Function Documentation

##### 10.24.2.a `def handlePrint (self):`

Connect to the printer and provide option for user printing :

- `fee__print.py`

##### 10.24.2.b `def handlePreview (self):`

Generate pdf preview for user with other function. :

- `fee__print.py`

##### 10.24.2.c `def handleTextChanged (self):`

enable button when text has been changed. :

- `fee__print.py`

## 10.25 fees Reference

### Public Member Functions

- `def __init__ (self):`
- `def show_information (self):`
- `def add (self):`
- `def update (self):`
- `def remove (self):`
- `def refresh_list (self):`
- `def conn (self):`

#### 10.25.1 Constructor & Destructor Documentation

##### 10.25.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

## 10.25.2 Member Function Documentation

### 10.25.2.a def show\_information (self):

Show specific fee information for selected item. :

- fees.py

### 10.25.2.b def add (self):

Open a new window for new entering new fee information.

:

- fees.py

### 10.25.2.c def update (self):

Update a fee information for user selected category. :

- fees.py

### 10.25.2.d def remove (self):

Remove a fee information for user selected category. :

- fees.py

### 10.25.2.e def refresh\_list (self):

Refresh the list of fees and populate to table view. :

- fees.py

### 10.25.2.f def conn (self):

Set up a connection with database. :

- fees.py

## 10.26 login Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def handleLogin (self):
- def getAccessLevel (self):
- def getUsername (self):
- def main (self):
- def conn (self):

## 10.26.1 Constructor & Destructor Documentation

### 10.26.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface. Have two fields with username entering and password entering.



## 10.26.2 Member Function Documentation

### 10.26.2.a def handleLogin (self):

Check weather username and password from user entered validity and then show appropriate message :

- login\_window.py

### 10.26.2.b def getAccessLevel (self):

Get the access level of current logged user for later usage. :

- login\_window.py

### 10.26.2.c def getUsername (self):

Get the username of current logged user for later usage :

- login\_window.py

### 10.26.2.d def conn (self):

Set up a connection with database. :

- login\_window.py

## 10.27 My info Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def check\_existing\_address (self):
- def fill\_form (self):
- def submit\_updates (self):
- def conn (self):

## 10.27.1 Constructor & Destructor Documentation

### 10.27.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

## 10.27.2 Member Function Documentation

### 10.27.2.a def check\_existing\_address (self):

Check weather it has existed address in the database if it has let user choose weather they want to add new address or update existing address. :

- My\_Information.py

### 10.27.2.b def fill\_form (self):

Populate the window with current teacher's information. :

- My\_Information.py

### 10.27.2.c `def submit_updates (self):`

Submit current information from window to database while doing error checking for each field. :

- `My_Information.py`

### 10.27.2.d `def conn (self):`

Set up a connection with database. :

- `My_Information.py`

## 10.28 Navi Reference

### Public Member Functions

- `def __init__ (self):`
- `def is_valid (self):`

### 10.28.1 Constructor & Destructor Documentation

#### 10.28.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, it shows user with two different authentication levels, admin and teacher.

### 10.28.2 Member Function Documentation

#### 10.28.2.a `def is_valid (self):`

Check weather user has right to enter corresponding page. :

- `Navi_dialog.py`

## 10.29 Partial pay Reference

### Public Member Functions

- `def __init__ (self):`
- `def pay_money (self):`
- `def get_rate (self):`
- `def select_student (self):`
- `def search_student (self):`
- `def conn (self):`

### 10.29.1 Constructor & Destructor Documentation

#### 10.29.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the list of students who owe money.

### 10.29.2 Member Function Documentation

#### 10.29.2.a `def pay_money (self):`

Open a new window for user check entering. :

- `partial_main_window.py`

**10.29.2.b def get\_\_rate (self):**

Get rate according to the amount of time student put onto class. :

- partial\_\_main\_\_window.py

**10.29.2.c def select\_\_student (self):**

Enable the button when user click on the window. :

- partial\_\_main\_\_window.py

**10.29.2.d def search\_\_student (self):**

Search student by their name and populate the result to table view. :

- partial\_\_main\_\_window.py

**10.29.2.e def conn (self):**

Set up a connection with database. :

- partial\_\_main\_\_window.py

## 10.30 Part pay dialog Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def pay (self):

### 10.30.1 Constructor & Destructor Documentation

**10.30.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database and populate the rate in combo Box.

### 10.30.2 Member Function Documentation

**10.30.2.a def pay (self):**

Make a payment with specified amount of money and method. :

- partial\_\_pay\_\_dialog.py

## 10.31 refund Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def getDropDate (self):
- def issueRefund (self):
- def getStudentId (self):
- def costofClass (self):
- def conn (self):

### 10.31.1 Constructor & Destructor Documentation

#### 10.31.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.31.2 Member Function Documentation

#### 10.31.2.a `def getDropDate (self):`

Open a new dialog and let input the drop date then reads back to the main window. :

- Add\_a\_Class.py

#### 10.31.2.b `def issueRefund (self):`

Check weather student can be refunded. :

- Add\_a\_Class.py

#### 10.31.2.c `def costofClass (self):`

get different discount scheme. :

- Add\_a\_Class.py

#### 10.31.2.d `def getStudentId (self):`

Get current selected student. :

- Add\_a\_Class.py

#### 10.31.2.e `def conn (self):`

Set up a connection with database. :

- Add\_a\_Class.py

## 10.32 removeAdmin Reference

### Public Member Functions

- `def __init__ (self):`
- `def submit (self):`
- `def conn (self):`

### 10.32.1 Constructor & Destructor Documentation

#### 10.32.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.32.2 Member Function Documentation

#### 10.32.2.a `def submit (self):`

Submit the remove request of a admin and then return to user appropriate message. :

- removeAdmin.py

**10.32.2.b def conn (self):**

Set up a connection with database. :

- removeAdmin.py

**10.33 removeClass Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def submit (self):
- def conn (self):

**10.33.1 Constructor & Destructor Documentation****10.33.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database.

**10.33.2 Member Function Documentation****10.33.2.a def submit (self):**

Submit the remove request of a Class and then return to user appropriate message. :

- removeClass.py

**10.33.2.b def conn (self):**

Set up a connection with database. :

- removeClass.py

**10.34 removeStudentData Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def refreshModel (self):
- def conn (self):

**10.34.1 Constructor & Destructor Documentation****10.34.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database.

**10.34.2 Member Function Documentation****10.34.2.a def refreshModel (self):**

Refresh the list of students in table view. :

- removeStudentData.py

### 10.34.2.b def conn (self):

Set up a connection with database. :

- removeStudentData.py

## 10.35 removeTeacher Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def submit (self):
- def conn (self):

### 10.35.1 Constructor & Destructor Documentation

#### 10.35.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database.

### 10.35.2 Member Function Documentation

#### 10.35.2.a def submit (self):

Submit the remove request of a Teacher and then return to user appropriate message. :

- removeTeacher.py

#### 10.35.2.b def conn (self):

Set up a connection with database. :

- removeTeacher.py

## 10.36 Roll Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def print\_student (self):
- def show\_class (self):
- def conn (self):

### 10.36.1 Constructor & Destructor Documentation

#### 10.36.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database. Populate the list of class of logged teacher and they can choose specific class to print out the roll sheet.

### 10.36.2 Member Function Documentation

#### 10.36.2.a def print\_student (self):

Open a new window and fill the window with the information generated based on class registration in html. :

- Role\_window.py

**10.36.2.b def show\_\_class (self):**

Show the classes that teacher are teaching currently. :

- Role\_\_window.py

**10.36.2.c def conn (self):**

Set up a connection with database. :

- Role\_\_window.py

**10.37 Print Roll Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def handlePrint (self):
- def handlePreview (self):
- def handleTextChanged (self):

**10.37.1 Constructor & Destructor Documentation****10.37.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, populate html information onto the window.

**10.37.2 Member Function Documentation****10.37.2.a def handlePrint (self):**

Connect to the printer and provide option for user printing :

- Role\_\_print.py

**10.37.2.b def handlePreview (self):**

Generate pdf preview for user with other function. :

- Role\_\_print.py

**10.37.2.c def handleTextChanged (self):**

enable button when text has been changed. :

- Role\_\_print.py

**10.38 schedule print Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def handlePrint (self):
- def handlePreview (self):
- def handleTextChanged (self):

### 10.38.1 Constructor & Destructor Documentation

#### 10.38.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, populate html information onto the window.

### 10.38.2 Member Function Documentation

#### 10.38.2.a `def handlePrint (self):`

Connect to the printer and provide option for user printing :

- `schedule_print.py`

#### 10.38.2.b `def handlePreview (self):`

Generate pdf preview for user with other function. :

- `schedule_print.py`

#### 10.38.2.c `def handleTextChanged (self):`

enable button when text has been changed. :

- `schedule_print.py`

## 10.39 Search class Reference

### Public Member Functions

- `def __init__ (self):`
- `def add_location (self):`
- `def Classinfo_update (self):`
- `def detail_show (self):`
- `def advsearch_show (self):`
- `def advsearch_query (self):`
- `def reset_table (self):`
- `def search (self):`
- `def conn (self):`

### 10.39.1 Constructor & Destructor Documentation

#### 10.39.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database. It shows a window with a table including classes' info and search fields and detail button.

### 10.39.2 Member Function Documentation

#### 10.39.2.a `def add_location (self):`

Add new location for given class. :

- `Search_class_window.py`

#### 10.39.2.b `def Classinfo_update (self):`

Update classes info such as name, address while doing error check based on regular expression. :

- `Search_class_window.py`



**10.39.2.c def detail\_show (self):**

Show detail of a single class such as name, address onto dialog. :

- Search\_class\_window.py

**10.39.2.d def advsearch\_show (self):**

This will open a advanced search dialog for further searching strategies. :

- Search\_class\_window.py

**10.39.2.e def advsearch\_query (self):**

This dialog will reads info from each fields and filter those word simultaneous and display them in table view. :

- Search\_class\_window.py

**10.39.2.f def reset\_table (self):**

Reset the list of classes in table view . :

- Search\_class\_window.py

**10.39.2.g def search (self):**

User can enter name in text filed and then search specific class by their name. :

- Search\_class\_window.py

**10.39.2.h def conn (self):**

Set up a connection with database. :

- Search\_class\_window.py

**10.40 Search teacher Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def Teacherinfo\_update (self):
- def detail\_show (self):
- def advsearch\_show (self):
- def advsearch\_query (self):
- def reset\_table (self):
- def search (self):
- def conn (self):

**10.40.1 Constructor & Destructor Documentation****10.40.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database. It shows a window with a table including teacher info and search fields and detail button.

## 10.40.2 Member Function Documentation

### 10.40.2.a def Teacherinfo\_\_update (self):

Update teacher info such as name, address, phone while doing error check based on regular expression. :

- Search\_teacher\_window.py

### 10.40.2.b def detail\_\_show (self):

Show detail of a single teacher such as name, address, phone onto dialog. :

- Search\_teacher\_window.py

### 10.40.2.c def advsearch\_\_show (self):

This will open a advanced search dialog for further searching strategies. :

- Search\_teacher\_window.py

### 10.40.2.d def advsearch\_\_query (self):

This dialog will reads info from each fields and filter those word simultaneous and display them in table view. :

- Search\_teacher\_window.py

### 10.40.2.e def reset\_\_table (self):

Reset the list of classes in table view . :

- Search\_teacher\_window.py

### 10.40.2.f def search (self):

User can enter name in text filed and then search specific teacher by their name. :

- Search\_teacher\_window.py

### 10.40.2.g def conn (self):

Set up a connection with database. :

- Search\_teacher\_window.py

## 10.41 Search student Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def stuinfo\_\_update (self):
- def detail\_\_show (self):
- def advsearch\_\_show (self):
- def advsearch\_\_query (self):
- def reset\_\_table (self):
- def search (self):
- def conn (self):

### 10.41.1 Constructor & Destructor Documentation

#### 10.41.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database. It shows a window with a table including student info and search fields and detail button.

### 10.41.2 Member Function Documentation

#### 10.41.2.a `def stuinfo_update (self):`

Update student info such as name, address, phone while doing error check based on regular expression. :

- Search\_window.py

#### 10.41.2.b `def detail_show (self):`

Show detail of a single student such as name, address, phone onto dialog. :

- Search\_window.py

#### 10.41.2.c `def advsearch_show (self):`

This will open a advanced search dialog for further searching strategies. :

- Search\_window.py

#### 10.41.2.d `def advsearch_query (self):`

This dialog will reads info from each fields and filter those word simultaneous and display them in table view. :

- Search\_window.py

#### 10.41.2.e `def reset_table (self):`

Reset the list of classes in table view . :

- Search\_window.py

#### 10.41.2.f `def search (self):`

User can enter name in text filed and then search specific student by their name. :

- Search\_window.py

#### 10.41.2.g `def conn (self):`

Set up a connection with database. :

- Search\_window.py

## 10.42 Update teacher Reference

### Public Member Functions

- `def __init__ (self):`
- `def checkExistingAddress (self):`
- `def clearForm (self):`
- `def fillForm (self):`
- `def submitUpdates (self):`
- `def conn (self):`

### 10.42.1 Constructor & Destructor Documentation

#### 10.42.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.42.2 Member Function Documentation

#### 10.42.2.a `def checkExistingAddress (self):`

Check weather entered address has been already stored in database if so remind user to add new or update.  
:

- `Select_teacher_dialog.py`

#### 10.42.2.b `def clearForm (self):`

clear information in text field on dialog :

- `Select_teacher_dialog.py`

#### 10.42.2.c `def fillForm (self):`

Fill out information of selected teacher onto dialog. :

- `Select_teacher_dialog.py`

#### 10.42.2.d `def submitUpdates (self):`

Submit information from text fields while doing error checking based on regular expression. :

- `Select_teacher_dialog.py`

#### 10.42.2.e `def conn (self):`

Set up a connection with database. :

- `Select_teacher_dialog.py`

## 10.43 Set semester Reference

### Public Member Functions

- `def __init__ (self):`
- `def add (self):`
- `def set (self):`
- `def conn (self):`

### 10.43.1 Constructor & Destructor Documentation

#### 10.43.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.43.2 Member Function Documentation

#### 10.43.2.a def add (self):

Add a new semester to the system and write it to the database. :

- set\_dialog.py

#### 10.43.2.b def set (self):

Set current semester to a existed semester. :

- set\_dialog.py

#### 10.43.2.c def conn (self):

Set up a connection with database. :

- set\_dialog.py

## 10.44 Show hours Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def update\_text (self):
- def update\_hours (self):

### 10.44.1 Constructor & Destructor Documentation

#### 10.44.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database. Allows teacher to enter their hours and calculate the final fee.

### 10.44.2 Member Function Documentation

#### 10.44.2.a def update\_text (self):

Get current rate according to the text selected in the combo box. :

- show\_hours\_dialog.py

#### 10.44.2.b def update\_hours (self):

Update the amount of hours by selected text fields :

- show\_hours\_dialog.py

## 10.45 Stu add Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def add\_stu (self):
- def set\_guradian (self):
- def add\_guradian (self):
- def show\_list (self):

### 10.45.1 Constructor & Destructor Documentation

#### 10.45.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. This allows user to enter info of student into multiple fields.

### 10.45.2 Member Function Documentation

#### 10.45.2.a `def add__stu (self):`

User can enter info of student into multiple fields and submit it to database. The window also does error checking for inappropriate info.

- `Stu_add_reg_dialog.py`

#### 10.45.2.b `def set__guradian (self):`

Set guardians of student and assign the guardians' id to them. the default id is 0 which initially assign to them. the function also contains all the error checking mechanism based on regular expression. :

- `Stu_add_reg_dialog.py`

#### 10.45.2.c `def add__guradian (self):`

This window allows user to enter the info of each guardian, while doing error checking and it also let user to update the info of existing guardians. :

- `Stu_add_reg_dialog.py`

#### 10.45.2.d `def show__list (self):`

It allows user to open a new window in order to register the classes, also giving them ability to drop the classes. :

- `Stu_add_reg_dialog.py`

## 10.46 Stu reg Reference

### Public Member Functions

- `def __init__ (self):`

### 10.46.1 Constructor & Destructor Documentation

#### 10.46.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. This window is facilitating the user registration.

### 10.46.2 Member Function Documentation

## 10.47 Stu pay Reference

### Public Member Functions

- `def __init__ (self):`
- `def get__rate (self):`

- `def search__student (self):`
- `def print__history (self):`
- `def select__student (self):`
- `def conn (self):`

### 10.47.1 Constructor & Destructor Documentation

#### 10.47.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.47.2 Member Function Documentation

#### 10.47.2.a `def get__rate (self):`

Get the corresponding rate of the student and then cache them into dictionary. :

- `student__owe__window.py`

#### 10.47.2.b `def search__student (self):`

Search student by their name which is entered by user. :

- `student__owe__window.py`

#### 10.47.2.c `def print__history (self):`

It will open a new diglog and print out the history of payment, user can preview it or print it by printer.  
:

- `student__owe__window.py`

#### 10.47.2.d `def select__student (self):`

it will get the student name and write into a global variable. :

- `student__owe__window.py`

#### 10.47.2.e `def conn (self):`

Set up a connection with database. :

- `student__owe__window.py`

## 10.48 Stu reg window Reference

### Public Member Functions

- `def __init__ (self):`
- `def refresh (self):`
- `def stu__add (self):`
- `def search__Stu (self):`
- `def stu__info (self):`
- `def select__Stu (self):`
- `def conn (self):`

## 10.48.1 Constructor & Destructor Documentation

### 10.48.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. User can choose specific student in table view and register classes for them.

## 10.48.2 Member Function Documentation

### 10.48.2.a `def refresh (self):`

refresh the list of student in table view in order to show student after adding new student. :

- Student\_reg\_window.py

### 10.48.2.b `def stu_add (self):`

Open a new window for adding new student. :

- Student\_reg\_window.py

### 10.48.2.c `def search_Stu (self):`

Search student by their name and show the result in the table view. :

- Student\_reg\_window.py

### 10.48.2.d `def select_Stu (self):`

Get the name of student when user clicked on table view. :

- Student\_reg\_window.py

### 10.48.2.e `def conn (self):`

Set up a connection with database. :

- Student\_reg\_window.py

## 10.49 Student schedule Reference

### Public Member Functions

- `def __init__ (self):`
- `def search_student (self):`
- `def print_schedule (self):`
- `def select_Student (self):`
- `def conn (self):`

## 10.49.1 Constructor & Destructor Documentation

### 10.49.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.



### 10.49.2 Member Function Documentation

#### 10.49.2.a def search\_student (self):

Search student by their name and populate the result in table view. :

- student\_schedule\_window.py

#### 10.49.2.b def print\_schedule (self):

Open a new dialog and print out the schedule in that window by using html. User can preview the content or print it out to printer.

:

- student\_schedule\_window.py

#### 10.49.2.c def select\_Student (self):

Get the student name when user clicked on them in table view. :

- student\_schedule\_window.py

#### 10.49.2.d def conn (self):

Set up a connection with database. :

- student\_schedule\_window.py

## 10.50 Teacher Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def logout (self):
- def enter\_hour (self):
- def reset\_user (self):
- def reset\_password (self):
- def update\_teacher (self):
- def roll\_sheet (self):
- def search\_class (self):
- def See\_teacher\_schedule (self):
- def Add\_student\_class (self):
- def See\_student\_schedule (self):
- def search\_student (self):
- def change\_window (self):

### 10.50.1 Constructor & Destructor Documentation

#### 10.50.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. This is a teacher landing page.

### 10.50.2 Member Function Documentation

#### 10.50.2.a def logout (self):

User can log out system by clicking logout button. :

- Teacher.py

**10.50.2.b def enter\_hour (self):**

Open a new window that teacher can enter their hours for different categories. :

- Teacher.py

**10.50.2.c def reset\_\_user (self):**

Open a new window that help them change their username. :

- Teacher.py

**10.50.2.d def reset\_\_password (self):**

Open a new window that help them change their password. :

- Teacher.py

**10.50.2.e def update\_\_teacher (self):**

Open a new window that teacher can change their personal info. :

- Teacher.py

**10.50.2.f def roll\_\_sheet (self):**

Open a new window that give teacher a ability to print out roll sheet of each class they teach. :

- Teacher.py

**10.50.2.g def search\_\_class (self):**

Open a window that teacher can search class info. :

- Teacher.py

**10.50.2.h def See\_\_teacher\_\_schedule (self):**

Open a new window that teacher can see their own schedule. :

- Teacher.py

**10.50.2.i def Add\_\_student\_\_class (self):**

Open a window that teacher can approve class. :

- Teacher.py

**10.50.2.j def See\_\_student\_\_schedule (self):**

Open a new window that teacher can see student schedule. :

- Teacher.py

**10.50.2.k def search\_\_student (self):**

Open a new window that teacher can search student. :

- Teacher.py

**10.50.2.1 def change\_\_window (self):**

change page. :

- Teacher.py

**10.51 Teacher history dialog Reference****Public Member Functions**

- def \_\_init\_\_ (self):

**10.51.1 Constructor & Destructor Documentation****10.51.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. It shows the classes teacher had taught previous semesters.

**10.51.2 Member Function Documentation****10.52 Teacher history window Reference****Public Member Functions**

- def \_\_init\_\_ (self):
- def search\_\_teacher (self):
- def print\_\_history (self):
- def select\_\_teacher (self):

**10.52.1 Constructor & Destructor Documentation****10.52.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

**10.52.2 Member Function Documentation****10.52.2.a def search\_\_teacher (self):**

Search teachers by their name and then populate the result to the table view. :

- teacher\_\_history\_\_window.py

**10.52.2.b def print\_\_history (self):**

Print teacher's teaching history in a new window. :

- teacher\_\_history\_\_window.py

**10.52.2.c def select\_\_teacher (self):**

Get teacher name when user clicked on table view. :

- teacher\_\_history\_\_window.py

## 10.53 Teacher payrate dialog Reference

### Public Member Functions

- `def __init__ (self):`
- `def update_text (self):`
- `def update (self):`
- `def insert (self):`

#### 10.53.1 Constructor & Destructor Documentation

##### 10.53.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box. Teacher can look into a specific rate info.

#### 10.53.2 Member Function Documentation

##### 10.53.2.a `def update_text (self):`

Get text info when user choose different combo box. :

- `teacher_payrate_dialog.py`

##### 10.53.2.b `def update (self):`

Update payrate for themself. :

- `teacher_payrate_dialog.py`

##### 10.53.2.c `def insert (self):`

Add a new payrate to database. :

- `teacher_payrate_dialog.py`

## 10.54 Teacher payrate Reference

### Public Member Functions

- `def __init__ (self):`
- `def search_teacher (self):`
- `def print_history (self):`
- `def select_teacher (self):`
- `def conn (self):`

#### 10.54.1 Constructor & Destructor Documentation

##### 10.54.1.a `def __init__ (self):`

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

## 10.54.2 Member Function Documentation

### 10.54.2.a def search\_teacher (self):

Search teacher by their name and populate the list to the table view. :

- teacher\_payrate\_window.py

### 10.54.2.b def print\_history (self):

Open a new window that can print out the payment history. :

- teacher\_payrate\_window.py

### 10.54.2.c def select\_teacher (self):

Get the teacher name when user click on the table view. :

- teacher\_payrate\_window.py

### 10.54.2.d def conn (self):

Set up a connection with database. :

- teacher\_payrate\_window.py

## 10.55 Teacher schedule Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def search\_student (self):
- def print\_schedule (self):
- def select\_Student (self):
- def conn (self):

## 10.55.1 Constructor & Destructor Documentation

### 10.55.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

## 10.55.2 Member Function Documentation

### 10.55.2.a def search\_student (self):

Search teacher by their name and populate the result in table view. :

- teacher\_schedule\_window.py

### 10.55.2.b def print\_schedule (self):

Open a new dialog and print out the schedule in that window by using html. User can preview the content or print it out to printer. :

- teacher\_schedule\_window.py

### 10.55.2.c def select\_Student (self):

Get the teacher name when user clicked on them in table view. :

- teacher\_schedule\_window.py

### 10.55.2.d def conn (self):

Set up a connection with database. :

- teacher\_schedule\_window.py

## 10.56 Tuition Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def show\_information (self):
- def change\_min\_checkbox (self):
- def change\_hour\_checkbox (self):
- def add (self):
- def update (self):
- def refresh\_list (self):
- def update (self):
- def conn (self):

### 10.56.1 Constructor & Destructor Documentation

#### 10.56.1.a def \_\_init\_\_ (self):

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.56.2 Member Function Documentation

#### 10.56.2.a def show\_information (self):

See the tuition rate. :

- tuition.py

#### 10.56.2.b def change\_min\_checkbox (self):

Set hourly or min. :

- tuition.py

#### 10.56.2.c def change\_hour\_checkbox (self):

Set hourly or min. :

- tuition.py

#### 10.56.2.d def add (self):

open a new window that add a new rate. :

- tuition.py

**10.56.2.e def update (self):**

Update the existing tuition rate. :

- tuition.py

**10.56.2.f def remove (self):**

Remove a existing tuition rate. :

- tuition.py

**10.56.2.g def refresh\_\_list (self):**

Refresh the list of tuition. :

- tuition.py

**10.56.2.h def conn (self):**

Set up a connection with database. :

- tuition.py

## 10.57 Update fees Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def change\_flat (self):
- def change\_percent (self):
- def fill\_window (self):
- def submit\_updates (self):
- def add\_rate (self):

### 10.57.1 Constructor & Destructor Documentation

**10.57.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.57.2 Member Function Documentation

**10.57.2.a def change\_flat (self):**

Change the flat rate. :

- Update\_fees\_window.py

**10.57.2.b def change\_percent (self):**

change the percentage. :

- Update\_fees\_window.py

**10.57.2.c def fill\_window (self):**

populate the window. :

- Update\_fees\_window.py

**10.57.2.d def submit\_updates (self):**

Submit the update. :

- Update\_fees\_window.py

**10.57.2.e def add\_rate (self):**

Add the new rate. :

- Update\_fees\_window.py

## 10.58 Update rates Reference

### Public Member Functions

- def \_\_init\_\_ (self):
- def setup\_database (self):
- def add\_location (self):
- def add\_class (self):
- def conn (self):

### 10.58.1 Constructor & Destructor Documentation

**10.58.1.a def \_\_init\_\_ (self):**

In the constructor, it is established an UI interface, then it is established a connection with database and populate the location in combo Box.

### 10.58.2 Member Function Documentation

**10.58.2.a def change\_flat (self):**

Change the flat rate. :

- Update\_rates\_window.py

**10.58.2.b def change\_percent (self):**

change the percentage. :

- Update\_rates\_window.py

**10.58.2.c def fill\_window (self):**

populate the window. :

- Update\_rates\_window.py

**10.58.2.d def submit\_updates (self):**

Submit the update. :

- Update\_rates\_window.py



**10.58.2.e** `def add_rate (self):`

Add the new rate. :

- `Update_rates_window.py`



---

## Bibliography

---

- [1] R. Arkin. *Governing Lethal Behavior in Autonomous Robots*. Taylor & Francis, 2009.
- [2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.
- [3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at <http://planning.cs.uiuc.edu/>.
- [4] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automation moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, pages 403–430, 1987.
- [5] S.A. NOLFI and D.A. FLOREANO. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. A Bradford book. A BRADFORD BOOK/THE MIT PRESS, 2000.
- [6] Wikipedia. Asimo — Wikipedia, the free encyclopedia. [http://upload.wikimedia.org/wikipedia/commons/thumb/0/05/HONDA\\_ASIMO.jpg/450px-HONDA\\_ASIMO.jpg](http://upload.wikimedia.org/wikipedia/commons/thumb/0/05/HONDA_ASIMO.jpg/450px-HONDA_ASIMO.jpg), 2013. [Online; accessed June 23, 2013].



# **SDSMT SENIOR DESIGN SOFTWARE DEVELOPMENT AGREEMENT**

This Software Development Agreement (the "Agreement") is made between the SDSMT Computer Science Senior Design Team DanceSoft

Consisting of team members Dicheng Wu and Marcus Berger,  
and Sponsor Jeff McGough,

with address: 501 E St Joseph St, Rapid City, SD 57701

## **1 RECITALS**

1. Sponsor desires Senior Design Team to develop software for use by the Academy of Dance Arts in South Dakota.
2. Senior Design Teams willing to develop such Software.

NOW, THEREFORE, in consideration of the mutual covenants and promises herein contained, the Team and Sponsor agree as follows:

## **2 EFFECTIVE DATE**

This Agreement shall be effective as of October 1, 2015

## **3 DEFINITIONS**

1. "Software" shall mean the computer programs in machine readable object code form and any subsequent error corrections or updates supplied to Sponsor by Senior Design Team pursuant to this Agreement.
2. "Acceptance Criteria" means the written technical and operational performance and functional criteria and documentation standards set out in the DanceSoft Senior Design Documentation
3. "Acceptance Date" means the date of the South Dakota School of Mines and Technology Senior Design Fair, or when all Deliverables have been accepted by Sponsor in accordance with the Acceptance Criteria and this Agreement.
4. "Deliverable" means a deliverable specified in the DanceSoft Senior Design Documentation section 1.3
5. "Delivery Date" shall mean, the end of the spring 2016 semester on which University has delivered to Sponsor all of the Deliverables in accordance with section 1.3 of the DanceSoft Senior Design Documentation and this Agreement.

6. "Documentation" means the documents, manuals and written materials (including end-user manuals) referenced, indicated or described in the DanceSoft Senior Design Documentation or otherwise developed pursuant to this Agreement.
7. "Milestone" means the completion and delivery of all of the Deliverables or other events which are included or described in section 1.3 of the Dancesoft Senior Design Documentation scheduled for delivery and/or completion on a given target date; a Milestone will not be considered completed until the Acceptance Date has occurred with respect to all of the Deliverables for that Milestone.

## **4 DEVELOPMENT OF SOFTWARE**

1. Senior Design Team will use its best efforts to develop the Software described in the DanceSoft senior design documentation. The Software development will be under the direction of or his/her successors as mutually agreed to by the parties ("Team Lead") and will be conducted by the Team Lead. The Team will deliver the Software to the satisfaction of the course instructor that reasonable effort has been made to address the needs of the client. The Team understands that failure to deliver the Software is grounds for failing the course.
2. Sponsor understands that the Senior Design course's mission is education and advancement of knowledge, and, consequently, the development of Software must further that mission. The Senior Design Course does not guarantee specific results or any results, and the Software will be developed only on a best efforts basis. The Software is considered a PROOF OF CONCEPT only and is NOT intended for commercial, medical, mission critical or industrial applications.
3. The Senior Design instructor will act as mediator between Sponsor and Team; and resolve any conflicts that may arise.

## **5 COMPENSATION**

No COMPENSATION will occur for this project.

## **6 CONSULTATION AND REPORTS**

1. Sponsor's designated representative for consultation and communications with the Team Lead shall be Jeff McGough or such other person as Sponsor may from time to time designate to the Team Lead.
2. During the Term of the Agreement, Sponsor's representatives may consult informally with course instructor regarding the project, both personally and by telephone. Access to work carried on in University facilities, if any, in the course of this Agreement shall be entirely under the control of University personnel but shall be made available on a reasonable basis.
3. The Team Lead will submit written progress reports. At the conclusion of this Agreement, the Team Lead shall submit a comprehensive final report in the form of the formal course documentation at the conclusion of the Senior Design II course.

## **7 CONFIDENTIAL INFORMATION**

1. The parties may wish, from time to time, in connection with work contemplated under this Agreement, to disclose confidential information to each other ("Confidential Information"). Each party will use reasonable efforts to prevent the disclosure of any of the other party's Confidential Information to third parties for a period of three (3) years after the termination of this Agreement, provided that the recipient party's obligation shall not apply to information that:
  - (a) is not disclosed in writing or reduced to writing and so marked with an appropriate confidentiality legend within thirty (30) days of disclosure;
  - (b) is already in the recipient party's possession at the time of disclosure thereof;
  - (c) is or later becomes part of the public domain through no fault of the recipient party;
  - (d) is received from a third party having no obligations of confidentiality to the disclosing party; (e) is independently developed by the recipient party; or (f) is required by law or regulation to be disclosed.
2. In the event that information is required to be disclosed pursuant to subsection (6), the party required to make disclosure shall notify the other to allow that party to assert whatever exclusions or exemptions may be available to it under such law or regulation.

## **8 INTELLECTUAL PROPERTY RIGHTS**

All intellectual property created for use in the DanceSoft project are covered under the PyQt GPL license. In accordance with the South Dakota Board of Regents and the GPL license the project will be provided to the client and the PyQt code must be made open source if the client chooses to distribute the DanceSoft project beyond the client's personal use.

## **9 WARRANTIES**

The Senior Design Team represents and warrants to Sponsor that:

1. The Software is the original work of the Senior Design Team in each and all aspects;
2. The Software and its use do not infringe any copyright or trade secret rights of any third party.

No agreements will be made beyond items (1) and (2).

## **10 INDEMNITY**

1. Sponsor is responsible for claims and damages, losses or expenses held against the Sponsor.
2. Sponsor shall indemnify and hold harmless the Senior Design Team, its affiliated companies and the officers, agents, directors and employees of the same from any and all claims and damages, losses or expenses, including attorney's fees, caused by any negligent act of Sponsor or any of Sponsor's agents, employees, subcontractors, or suppliers.
3. NEITHER PARTY TO THIS AGREEMENT NOR THEIR AFFILIATED COMPANIES, NOR THE OFFICERS, AGENTS, STUDENTS AND EMPLOYEES OF ANY OF THE FOREGOING, SHALL BE LIABLE TO ANY OTHER PARTY HERETO IN ANY ACTION OR CLAIM FOR CONSEQUENTIAL OR SPECIAL DAMAGES, LOSS OF PROFITS, LOSS

OF OPPORTUNITY, LOSS OF PRODUCT OR LOSS OF USE, WHETHER THE ACTION IN WHICH RECOVERY OF DAMAGES IS SOUGHT IS BASED ON CONTRACT TORT (INCLUDING SOLE, CONCURRENT OR OTHER NEGLIGENCE AND STRICT LIABILITY), STATUTE OR OTHERWISE. TO THE EXTENT PERMITTED BY LAW, ANY STATUTORY REMEDIES WHICH ARE INCONSISTENT WITH THE PROVISIONS OF THESE TERMS ARE WAIVED.

## 11 INDEPENDENT CONTRACTOR

For the purposes of this Agreement and all services to be provided hereunder, the parties shall be, and shall be deemed to be, independent contractors and not agents or employees of the other party. Neither party shall have authority to make any statements, representations or commitments of any kind, or to take any action which shall be binding on the other party, except as may be expressly provided for herein or authorized in writing.

## 12 TERM AND TERMINATION

1. This Agreement shall commence on the Effective Date and extend until the end of classes of the second semester of Senior Design (CSC 464), unless sooner terminated in accordance with the provisions of this Section ("Term").
2. This Agreement may be terminated by the written agreement of both parties.
3. In the event that either party shall be in default of its materials obligations under this Agreement and shall fail to remedy such default within thirty (30) days after receipt of written notice thereof, this Agreement shall terminate upon expiration of the thirty (30) day period.
4. Any provisions of this Agreement which by their nature extend beyond termination shall survive such termination.

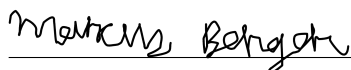
## 13 ATTACHMENTS

Attachments A and B are incorporated and made a part of this Agreement for all purposes.

## 14 GENERAL

1. This Agreement constitutes the entire and only agreement between the parties relating to the Senior Design Course, and all prior negotiations, representations, agreements and understandings are superseded hereby. No agreements altering or supplementing the terms hereof may be made except by means of a written document signed by the duly authorized representatives of the parties.
2. This Agreement shall be governed by, construed, and enforced in accordance with the internal laws of the State of South Dakota.

## 15 SIGNATURES



Marcus Berger

4/5/2016

Date



Dicheng Wu

4/5/16

Dicheng Wu

Date

Jeff McGough

4/5/16

Jeff McGough

Date



# A

---

## Product Description

---

The DanceSoft product developed by the South Dakota School of Mines and Technology senior design team of Marcus Berger and Dicheng Wu is a proof of concept for an administrative software to run the day to day activities of the Rapid City Academy of Dance Arts. The teams overall goal is to provide a proof of concept to the client Dr. Jeff McGough, to show that a viable software is indeed possible to meet the needs of the Academy and eventually through continued iteration by other senior design groups or by outside entities the project will replace the software currently in use at the Rapid City Academy of Dance Arts

The product will demonstrate a set of core functionalities that will be used to run the Academy, these functions include adding students, and teachers, assigning classes to teachers, registering students, ability to produce schedules and roll sheets, and a very rudimentary billing system that will be a blueprint for future iterations of the software. Other functions needed to complete this core functionality are also included in the system. The projects provides an admin interface where the teachers who are registered as admin can do several data manipulation task for the system. These task include managing classes, students and teachers, updating Academy personal information, and dealing with a simple billing system. The project also provided a teacher interface where teachers registered in the system are able to manage their needs. Teachers are able to view student information such as name, guardian, address, and emergency contact. Teachers are also able to search classes, produce student and teacher schedules, and produce class role sheets. Lastly teachers are able to modify their personal information, this includes a form to change fields such as address, and medical information, and the user can also modify name, date of birth and other field should mistakes be made when the teacher is first entered. Also the teacher is able to change their log in credentials and enter their hours for viewing by system admins.

As a proof of concept this software is meant to give the client a minimum viable product, or more likely provide a blueprint for a future system iteration whether that be a future senior design project, or a contract software. In accordance with the clients request this product is not meant to be an industrial software at the completion of the senior design project. The overall goal remains to give the client an idea of the possibilities of a full industrial Academy of Dance Arts software.



# B

---

## Sprint Reports

---

### 1 Sprint Report #1

Sprint Report #1

#### 1.1 Team Members:

Dicheng Wu  
Marcus Berger

**Sponsor:**  
Jeff McGough

#### 1.2 Customer description

##### 1.2.a Description of sponsoring customer

The sponsoring customer for this project is Dr. Jeff McGough a computer science professor at the South Dakota School of Mines and Technology and the vice president of the Academy of Dance Arts in Rapid City, South Dakota. Well not the sponsor of the project Dr. McGough's wife is also a key part of the customer base as the owner of the Academy. She and other dance school owner are the target group for this project.

##### 1.2.b Statement of customer's problem or goal for this project

The customer wants a software that can run the day to day operations of a dance studio and also handle record keeping, billing, payroll, and other business operations

##### 1.2.c Customer's Needs

The customer needs us to develop a software solution which can run the dance studio in an effective manner. The product also needs to handle changing classes from year to year without needing to be updated. This means that the software needs to sync with multiple users, and handle new information such as class rosters, prices, clothing requirements for classes, changes in the employment roster, and many other changes that can occur in the running of a dance school.

This project as a whole needs to be an improvement on the current system in use by the customer and provide an easy and efficient way to run the clients business.

#### 1.3 Overview of the project:

The project consists of three major parts, GUI, database and back end code. For the Gui part, we are going to integrate everything into a small number of windows to eliminate the need for large numbers of window like the current product in use at the academy. and, thus, users can manipulate it very straightforwardly.

For database part, we are going to build a database which stores students, employees, classes, billing, and other information needed for the academy to function as a business. For the code back end, we are going to develop it on using Python and PyQt with a primary focus on Mac but with cross platform compatibles.

## **1.4 Project Environment:**

### **1.4.a Project boundaries**

The boundaries of this project would be the Academy of Dance Arts in Rapid City. The product could be used by other dance schools in the future but they are out of the scope of this projects development

### **1.4.b Project context**

The context of this project is only to develop a better software solution to the current software in use at the Academy of Dance Arts in Rapid City, South Dakota, so they can run their school in a more efficient manner. While the project is open source it is not the intention of this team to develop an all purpose solution for all the dance schools around the country. This project is tailored to the needs of the Academy of Dance Arts.

## **1.5 Project deliverables of Sprint 1:**

1. The research into program languages, database and GUI frameworks and architectures for the DanceSoft project.
2. Final decision on frameworks and architectures for the DanceSoft project.
3. User Stories and Product Backlogs for the project.
4. Creating a very simple Qt window.

## **1.6 User Stories**

After the requirement for the project were laid out the team created the user stories based on those requirement. The user stories the team came up with are as fallows:

1. As a user i want to adjust students payment models
2. As the owner I would like to see automatic database backups.
3. As a student I would like to be able to register online (with special app). Classes must be approved before added.
4. As a student I would like to be able to search clothing requirements.
5. As a student I would like to know my billing.
6. As the owner I would like to indicate clothing requirements per class.
7. As a studio person, I would like to be able to add students to classes.
8. As a student, teacher etc, I would like to be able to look up a students class list.
9. As the teacher I would like to get a class role for each class.
10. Given a class list, I would like to get an invoice of the tuition due.
11. Studio would like to track payments and estimate remainder due. I would like to generate an invoice for this amount.
12. As a student I would like to be able to register online (with special app). Classes must be approved before added.
13. As a student I would like to know my billing.

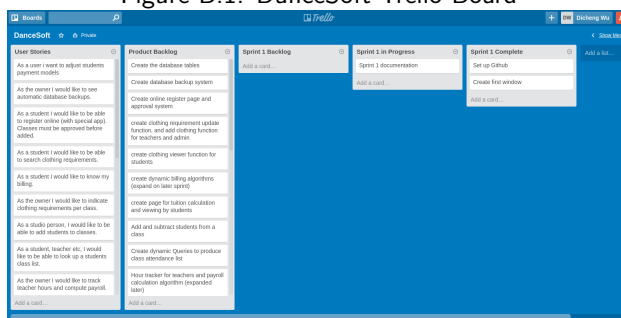
14. As the owner I would like to track teacher hours and compute payroll.
15. As the owner I would like to indicate clothing requirements per class.
16. As a student I would like to be able to search clothing requirements.
17. As the owner I would like to see automatic database backups.

## 1.7 Product Backlog:

From the above user stories the team produced the following product backlog. The parts are not in the order of execution. More thing will be add or removed to this in the future as the software develops.

1. Create the database tables
2. Create database backup system
3. Create online register page and approval system
4. create clothing requirement update function, and add clothing function for teachers and admin
5. create clothing viewer function for students
6. create dynamic billing algorithms (expand on later sprint)
7. create page for tuition calculation and viewing by students
8. Add and subtract students from a class
9. Create dynamic Queries to produce class attendance list
10. Hour tracker for teachers and payroll calculation algorithm (expanded later)
11. Using dynamic DNS service to set up Linux Box
12. Encrypt data that store in the database
13. retrieve data from database to produce a invoice for payroll
14. queries in database to retrieve student information
15. Create ability for employee to look up and modify student registration info
16. Query database to produce class role sheet
17. Query database to produce employee information
18. create permission assignment system
19. handle billing info (expand later)
20. Create dynamic algorithm for payment model creation and calculation.
21. Algorithm for payment tracking and remainder calculation, and query database to produce invoice.
22. Query and insert information needed to create and new class and produce a results page.
23. Create update query to assign teachers to a class

Figure B.1: DanceSoft Trello Board



### 1.7.a Sprint 1 Backlog:

1. Set up Github
2. Design Research
3. Design Decision
4. Create first window
5. Sprint 1 documentation
6. Sprint 1 Review
7. Continuing practice with QT

## 1.8 Research

### 1.8.a Database Research

### 1.8.b SQL VS NonSQL:

SQL is designed for fixing data structure and the scale of amount of data in database should be medium size otherwise with it growing big the database will drop down its speed. NonSQL is designed for frequent changing data structure and the scale of amount of data in database does not affect the performance of the database a lot. The SQL database is stable and however the NonSQL is constantly suffering fail overs. The database aims to store employees and students and classes information and the size of students less than 4000 and the size of employees less than 10. The structure of data is stable, by considering all those facts above, we decide to use SQL database.

#### The different SQL Databases:

For this part, we think of using MySQL. MySQL is a free software and widely used in different fields. It has a very good portability and can runs over different OS systems and aims for small size of data. Also our experience is more focused in MySQL, and the feature set for SQL in MySQL fits within the scope of the project, for these reasons we choose MySQL as the database frame work.

### 1.8.c Storage Options:

For this part, We considered three options, local, cloud, mixed. Since the database is accessed by different devices and the internet in the Academy is not stable, we decide to use mixed storage schema either Amazon AWS plus a local database or a Linux box plus a local database. After researching Amazon AWS, it does not support storing images and videos which the Academy may require in the future, therefore due to some inconsistency in the internet and the current and possible future needs of the Academy we decided to go with a local Linux box and database.



### 1.8.d Languages

Since the Academy is currently running a Mac system the first language idea was objective-C. Recently though Apple released a programming language update called Swift. Due to our teams inexperience with Mac and the possibility of the dance academy being sold in the future we were faced with a choice, between Python or Swift. Swift is a c++ like language which stuck out as a possible jumping off point for us. However as we researched it became clear that swift would have a high ramp up time and learning curve. On top of this as novices to the Mac development environment we would spend a large amount of time just trying to figure out the system. As far as pluses for Swift the language has all the functionality of objective-C plus things added to make the language better, overall reception for the language within the Mac community have been positive. The language itself is designed with porting to IOS in mind which would make a mobile transition more likely. The language would use Cocco as a GUI environment which is also relatively well revived by OSX developers, ut again ramp up for our team would be high.

The language competing with Swift in our discussions was Python. Python has the upside of being a language the team is more experienced in. Also the client Dr. McGough knows Python so any updates would be easier for him to do. Python is also a cross platform language which allows the team to produce working code for Mac, Windows, and Linux at the same time and on any operating system. This means the team can develop on Windows or Linux, development environments we are more familiar with, and have the code work on Mac. Also development in Python means that if the academy was ever sold to a Windows user the product would still work. Another advantage we discovered to python is the academic and career experience it provide since in our research for the SD Mines career fair we discovered that many companies use Python and more than expected would like QT experience. Lastly the Python community is larger and can more readily provide assistance if needed through websites and research.

So overall while Python is not Mac native it provides more viable reasons for use in our teams eyes. That is not to say we don't believe Swift is a good choice, Swift is a completely viable choice for a product like this it is just not the best for the exact situation the team is in. So we have decided to create the Academy of Dance Arts Software in Python using PyQt as a GUI.

## 1.9 Final Framework Decision

Language: Python

Gui: PyQt

Database: Mysql

### 1.10 Foreseeable issues

As of the sprint 1 review, possibly issues could include database encryption and synchronization, team learning curve of Qt.

## 2 Sprint Report #2

Sprint Report #2

### 2.1 Team Members:

Marcus Berger

Dicheng Wu

**Sponsor:**

Jeff McGough

## 2.2 Prototype Progress

The progress is mostly out of the research phase and has began the first steps of production. The progress is laid out in more detail in this report. As of now the prototype is in a state to begin working on functionality. The rough (not final) GUI pages are laid out and being constructed to give us an environment for creating the functionality. The back end database has been constructed and will allow us to begin testing the database connected page as we create them. Current pages constructed are log in page, landing pages, and some options page.

## 2.3 Project deliverables of Sprint 2:

1. The database creation script
2. Gui path work, meaning figuring out the path through the Gui architecture
3. Log in page that reads users permission level and sends them to the correct landing page
4. Rough landing pages for Admin and Teachers (design improvements to come in later sprint)

### 2.3.a Sprint 2 Backlog:

1. Creation of database tables
2. Draw GUI path
3. Decision on rough GUI theme
4. Create a log in page that sends the user to the correct landing page based on their permission level
5. Create rough versions of teacher and admin landing pages to test functionally (improve look in latter sprint)
6. Continue rough GUI page creation to have environments for functionality
7. Sprint 2 Review
8. Continuing practice with QT

## 2.4 Database Creation

The first thing we tackled in sprint 2 was getting the database up so we could begin actually developing the product and give our qt interface something to actually interface with. The main goal here was to make sure we had constructed the database in such a way that it could complete all the user stories it needed to in a way that made sense. After talking through the tables and the users stories we came up with the table creation script that was submitted with this sprint. While modifications will need to be made for when we do the billing side of the project, we think that this table structure should provide for the need of the academy.

A few thing to note in the database, we created student\_class and teacher\_class tables to deal with the many to many relationships in the database. Also there are no payroll or transaction tables yet as the group is still trying to figure out how to handle billing information and data.

## 2.5 GUI Work

### 2.5.a GUI Path and Theme

The path for the gui was part of the backlog so we could figure out the minimum number of pages we would need to do the things required by this project. This does not mean we are looking to create only the minimum but just get a general idea of how many pages we would need. Also it allowed us to see how the pages were connected and how we expect navigation through the system to work. The rough drawing of the path can be found in the Github repo's sprint two folder.

Figure B.2: Tables currently in database

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Data
Account	MyISAM	10	Dynamic	3	30	112.0 bytes	256.0 TB	5.0 KB	
Address	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Admin	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Class	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Guardian	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Student	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Student_Class	MyISAM	10	Fixed	0	0	0.0 bytes	4096.0 TB	1.0 KB	
Teacher	MyISAM	10	Dynamic	0	0	0.0 bytes	256.0 TB	1.0 KB	
Teacher_Class	MyISAM	10	Fixed	0	0	0.0 bytes	2304.0 TB	1.0 KB	

Figure B.3: Current iteration of the log in page

The other thing we need was to talk to the client about what he wanted for a navigation theme. We came up with three options, first was a button layout on the top and side of the page, second was drop down menu similar to the way was mac and windows handle navigation in their products. Last was an expanding folder structure similar to that of the content management system Ektron. After consulting with Dr. McGough, he decided that the more familiar drop down menu would be the easiest and most user friendly option and the option he would prefer we do. This decision allows us to use the built in menu bar functionality of PyQt and should reduce work load on navigation for this project.

Another part of theme is color and design, the color scheme will most likely resemble the academy's color plate, but the improve design will be worked on later. As of now the primary focus is making sure all the functionality works.

## 2.5.b GUI Pages

The GUI pages for this sprint were the log in page and the landing pages for admin and teachers. The student part of the gui will be php which will be worked on in a future sprint once the functionality is done. Since the student functions are similar to some of the admin or teacher functions most work should port over fairly quickly.

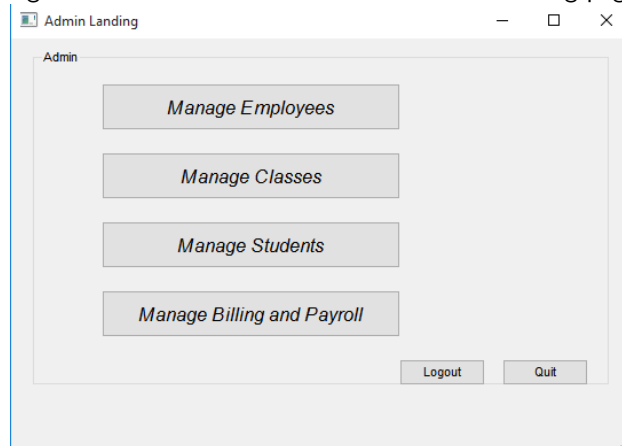
The first page we made was a log in page that takes a user name and a password and first checks to see if they exist and are correct within the system. If they are not a dialog saying whats wrong appears and prompts the user again. If the information passes the check the system reads the users permission level and sends them to the corresponding landing page.

The other set of pages we need to make to begin writing functionality was the landing pages for the admin and teacher permission levels. These pages show the types of things each can do and allow navigation to the various different functionality that permission level can do. For example the admin landing page has a button to take you to a student options page, from there you select which option you want and will be taken to the page to execute that function. The idea is to start at the landing page and be able to navigate to a specific function in three or four clicks max. The menu bar will also execute navigation and should allow the user to jump to a desired function.

Pages currently being worked on include:

1. Search pages

Figure B.4: Current iteration of the Admin Landing page



2. Modify data page
3. Add information and new information pages

## 2.6 Sprint 2 Issues

Some issues that were encountered during this sprint were:

1. Running out of time - the group found it hard to complete this sprint on time due to many time draining issues, such as other class work, family matters, midterm exams and a client presentation during this sprint. Mitigation for this issue will be better time management for the team.
2. Inexperience with PyQt - The team ran into some issues that drained on time when working with PyQt, such as looking up information on how to perform a needed task. Mitigation for this issue is mostly experience based as we learn the system and learn how to more efficiently navigate the community the time sink should go down.
3. Team communication - The team found some issues in communicating with one another because of the language barriers within the team. Which in some cases slowed sprint progress. Mitigation for this issue will come with time as the group begin to understand each other better and find efficient patterns of communication.

## 2.7 Client Interactions

Client interactions came in the form of weekly meeting every wednesday at 2:00 p.m. Topics included:

1. Progress reports on where we are in the project
2. Asking questions to refine functionality and understand academy needs
3. Discuss the future of the project and idea on how to execute them

## 2.8 Group Meeting

The group has a standard meeting time of 4:00-5:00 or 4:00-6:00 MWF and a second meeting time of TTH 10:00 - 12:00

Meeting can continue pass these times if needed, and other times during the weekend. However weekend times fluctuate and do not remain constant from week to week.

## 2.9 Work Distribution

Marcus:

1. GUI Path Charts
2. Landing page Construction

Dicheng:

1. GUI Theme Ideas
2. Log In Page and Permission Navigation

Together:

1. Wrote and talked through database and construction
2. Got database running
3. GUI page breakdown based on user stories and product backlog
4. Coded some functionality

## 3 Sprint Report #3

Sprint Report #3

### 3.1 Team Members:

Marcus Berger

Dicheng Wu

**Sponsor:**

Jeff McGough

### 3.2 Prototype Progress

The progress is being made on the simpler functions of the projects including search, updates, and role sheets. As of now the prototype is in a state of some working functionality on the admin and teacher side. The pages are not yet tied together but the complete functions function independently. Several GUI pages are complete or in a working state to compliment these functions. The back end database has been slightly updated to adjust to the needs of the project. Lastly the prototype has reached a state where testing can be conducted on parts of the project, more information is given later in this document.

### 3.3 Project deliverables of Sprint 3:

1. Student search page
2. Employee search page
3. Add a class to the database
4. Update teacher and student information pages
5. Modify clothing requirements for a class
6. Generate a class role sheet
7. Assign teacher to a class
8. Ability to add and subtract students from a class

Figure B.5: Search Pages

ID	Name	Gender	Date of Birth	Phone	Primary Guardian
1 1	Tom	m	2000-01-01	605-555-5555	Tom
2 2	Alice	f	2003-11-13	605-555-5555	Tim
3 3	Petra	f	2005-04-12	605-555-5555	Jim
4 4	Kate	f	2006-11-10	605-555-5555	Amy
5 5	Tyler	m	2003-02-22	605-555-5555	Lori

### 3.3.a Sprint 3 Backlog:

1. Queries in database to retrieve student and employee information
2. Query and insert information needed to create and new class and produce a results page.
3. Query database to produce class role sheet
4. Create clothing requirement update function, and add clothing function for teachers and admin
5. Create update query to assign teachers to a class
6. Create ability for employee to look up and modify student registration info
7. Add and subtract students from a class
8. Sprint 3 Review
9. Create Client presentation
10. Documentation for semester

### 3.4 Search Pages

The first page tackled this sprint was the search page. The page takes user input and searches based on name using a fuzzy search. Also included in both searches is an advanced search option that allows the user to check boxes corresponding to the fields in the database, which allows the user to search in more versatile ways. Lastly the user can click on a student to pull up all of their information and modify it as needed.

### 3.5 Add A Class

Clicking on the "Add a Class" button on the admin landing page will open a dialog. From this dialog box the user can type in information to add a class. These fields include class name, cost, start and end times, day, clothing requirements, class description, start and end dates, and the age range for the class. At the time of class creation only name is a required field. When the submit button is clicked a message box will pop up to confirm the database submission before submitting to the database.

### 3.6 Role Sheet

This page generates a list of classes teachers are currently teaching. Teacher can see who is taking his/her class by selecting the corresponding class on the list and can print this list as a pdf.

### 3.7 Update Pages

These pages are fairly simple to explain sometimes the user will need to add or alter a record of some kind these pages need to be able to do that in a simple and concise way. The update pages worked on in this sprint include student information from the employee side, updating teacher information and updates to clothing requirements for a class. Most of these page will just produce a form where information can be displayed and updated.

### 3.8 Assign Teacher to a Class

This page allows an admin to select a class, clicking on a class pulls up a list of available teachers to select to teach the class. Clicking on a teacher will pull up a message box to confirm the selections before submitting to the database. The available teachers are selected based on the start time of the selected class and the end time of the classes already being taught by each teacher. If the class times overlap then that teacher will not show up in the available teachers list.

### 3.9 Sprint 3 Issues

Some issues that were encountered during this sprint were:

1. Time - While not as much of a problem as it was in sprint two the group still found itself running low on time. The lead gained during Christmas break should reduce the issue in phase 2.
2. Inexperience with PyQt - While still a small issue the experience of the team is growing and while some issues still exist such as needing to look up information at times, the issues are lessened from the last sprint.
3. Team communication - The team found some issues in communicating with one another because of other projects and assignments. Which in some cases slowed sprint progress. Mitigation for this issue will be to more strongly impose a schedule in phase 2.

### 3.10 Client Interactions

Client interactions came in the form of weekly meeting every Wednesday at 2:00 p.m. Topics included:

1. Progress reports on where we are in the project
2. Asking questions to refine functionality and understand academy needs
3. Discuss the future of the project and idea on how to execute them

### 3.11 Group Meeting

The group has a standard meeting time of TTH 10:00 - 12:00 and once during the weekend usually from 1:00-5:00

Meeting can continue pass these times if needed, and other times during the weekend. However extra weekend times fluctuate and do not remain constant from week to week.

### 3.12 Work Distribution

Marcus:

1. Add a class
2. Assign teacher to class
3. Update teacher page
4. Update clothing

5. Documentation
6. Trello management

Dicheng:

1. Search Pages
2. Role sheet
3. Modify student information
4. Add and Subtract students from a class
5. Navigation to teacher or admin page

Together:

1. Updated database construction
2. GUI page breakdown based on user stories and product backlog
3. Coded some functionality

## 4 Sprint Report #4

Sprint Report #4

### 4.1 Team Members:

Marcus Berger

Dicheng Wu

**Sponsor:**

Jeff McGough

### 4.2 Prototype Progress

During sprint 3.5 the team organized the functionality and tied them together to create the single current working prototype. Other updates were made to fix known bugs and add a class search. While the team worked on putting the project together smaller bugs or needed pages that were implied by user stories were tackled as they came up. This resulted in too much time being used, so the student interface was not tackled. The student interface has since been dropped from the client's project requirements due to time.

Sprint 4 was dedicated to payroll and billing. First the client rewrote the user stories to provide more clarity to the team. After this the user stories were tackled by the group to produce the first draft of the payroll and billing interface. This included logging hours, payments, fees, rates, credits, and discounts. After this was done the team demoed the project for the client and got notes on the project as a whole. These notes will be tackled as part of the backlog in the next sprint.

### 4.3 Project deliverables of Sprint 3.5 and 4:

Sprint 3.5 had no defined deliverables, and was mostly clean up.

Sprint 4 deliverable was the first draft of the payroll and billing interfaces and functions, laid out in the user stories below.



#### **4.3.a Sprint 3.5 Backlog**

1. Fix Assign Teacher to Class Dialog Bug
2. Add Class Search
3. Complete Role Sheet
4. Tie Functions Together
5. Fix Smaller Bugs

#### **4.3.b Sprint 4 Backlog:**

1. View Teaching History
2. Look at The Current Amount Someone Owes
3. Generate an Invoice For The Amount Due
4. Look at Billing History
5. Apply Credits to a Students Account
6. Enter the Tuition and Fees Rate
7. Give Early Registration Discounts
8. Enter Staff Pay Rates
9. Enter a Full Payment for One Student
10. Enter a Full Payment for Several Students
11. Enter Payments From Multiple Sources For One or More Students
12. Compute Teacher Wages
13. Enter Staff Hours
14. Give Prorated Refunds

#### **4.4 Sprint 3.5**

During sprint 3.5 Class Search was added this page follows the same format as other search pages created in previous sprints. Also completed fixes to the assign teacher to class bug, role sheet completeness, and a few other minors bugs. At the end of the sprint the team tied what we had together so the product could run as a unit starting from the log in screen.

#### **4.5 Prorated Refunds**

The prorated refunds code will keep track of how much a class costed a student. If the student drops a class it will refund the remaining worth of the class to the students school credit field. This file is still in progress.

#### **4.6 Enter Staff Hours**

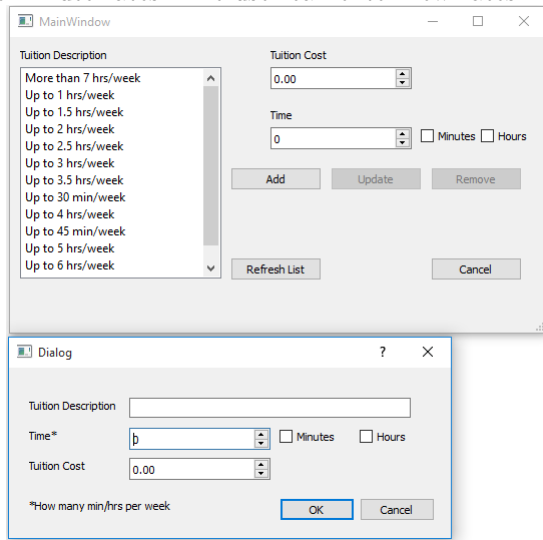
This page needs to undergo changes after meeting with client. The new version will allow a teacher to log their in class hours, office hours, trip hours, etc. These hours will then be paid out at their different rates respectfully. The hours will be logged as single numbers not by days or weeks.

## 4.7 Enter Teacher Wages

This was handled in the database in previous sprints assuming one pay rate. However in talks with the client the academy pays different pay rates based on what part of academy work they are doing. So the page will need to be updated once the list of possible pay rates have been provided by the client.

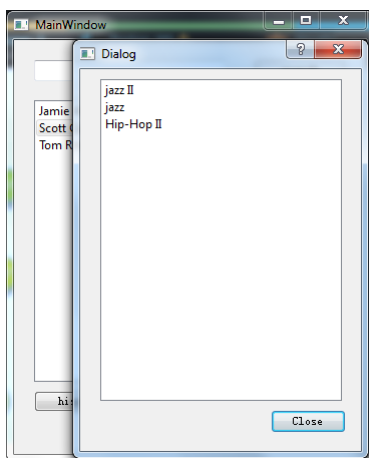
## 4.8 Enter Tuition Rates and Fees

These pages allow the user to look at the current tuition rates and fees. The user can then update existing rates, add new rates, or remove rates that are no longer needed. Tuition rates are logged in the database as flat minute rates. The user can enter new rates in the form of minutes or hours.



## 4.9 Teacher History

This page includes several components. Firstly, user can use this page to find a specific teacher by entering a full or partial name of teachers. Secondly, users can click the history button to open up another window which has a list of classes the teacher has taught.

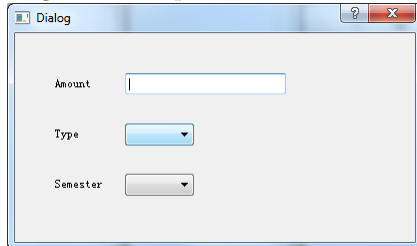


## 4.10 Set Semester

This page helps users to set the current semester from pool of semester and add a new semester to the system.

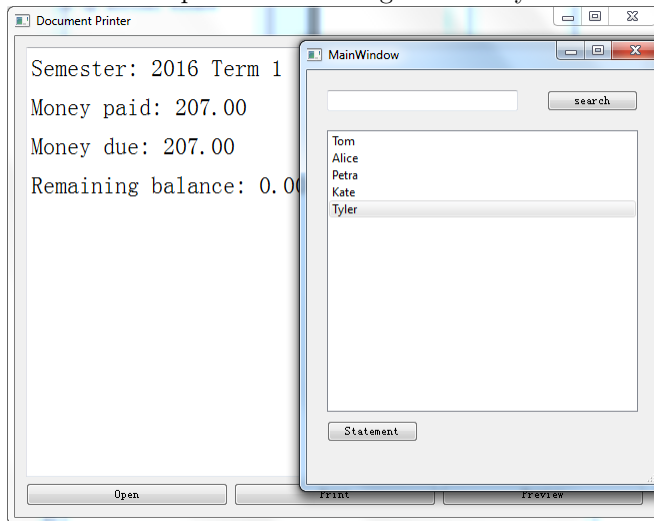
## 4.11 partial payment

This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the pay button to open up another window which asks user to input amount of money paid, payment's method and semester paid. The user also can choose single or multiple students at one time.



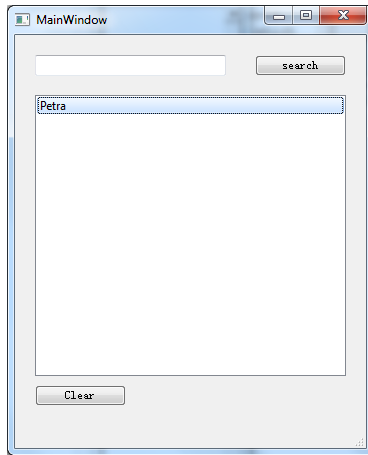
## 4.12 Student's owe

This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the statement button to open up another window which shows the amount of students paid, the amount of students due and the balance of student's account. Users also can print the invoice generated by the window.



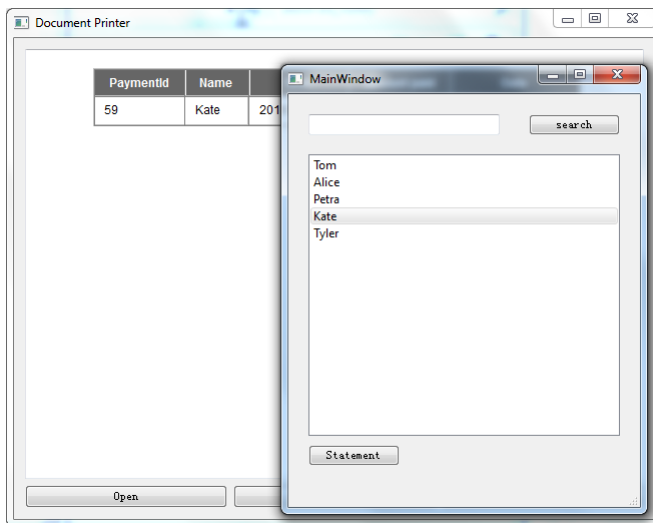
## 4.13 enter payment

This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the clear button to clear the due of selected students. Users can select single or multiple students at one time.



#### 4.14 bill history

This page includes several components. Firstly, user can use this page to find a specific student by entering a full or partial name of students. Secondly, users can click the statement button in order to get the list of payments. In addition, user can print the bill history.



#### 4.15 Sprint 3.5 and 4 Issues

Some issues that were encountered during this sprint were:

1. During winter break the team wanted to tackle the student interface app. Building other required pages in the main local GUI made this a problem. Therefore while some implied GUI pages were fixed and created, the team was unable to get to the student interface. As a result the student interface was removed from the project requirements.
2. Spring semester ramp up - Coming back from a month break meant that the team needed to ramp back up and get used to the new classes and new commitments. This caused the team to move slower than it should have in this sprint.
3. Team communication - The meetings this sprint were less effective as team members would have other commitments, or in some cases lack the drive to do the work. This lessened as the sprint continued, but sometimes still came up.

4. Requirement Confusion - At times during this sprint the requirement were confusing. This resulted in a rewrite of the requires for payroll and billing to provide more clarity and adjust for dropping the student interface. The rework helped the team greatly, however the user stories still sometimes left out core part of how the academy does its work compared to other places. This resulted in spending more then the usually time speaking with the client, and group friction.

#### **4.16 Client Interactions**

Client interactions came in the form of requires reviews and a demo of the prototype at the end of the sprint. Topics included:

1. Progress reports on where we are in the project
2. Asking questions to refine functionality, understand academy needs, and update requirements.
3. Discuss the future of the project and idea on how to execute them.
4. Prototype the current version of the prototype.

#### **4.17 Group Meeting**

The group has a standard meeting time of TTH 10:00 - 12:00 and once during the weekend and at home as needed.

Meeting can continue pass these times if needed, and other times during the weekend. However extra weekend times fluctuate and do not remain constant from week to week.

#### **4.18 Work Distribution**

Marcus:

1. 3.5 Fix Assign Teacher Bug
2. 3.5 Update GUI Functions, Minor Additions and Fixes
3. 3.5 Tie Various Functions Into a Single Prototype
4. Prorated Refund (Still in Progress)
5. Enter Teacher Hours (Still in Progress)
6. Teacher Pay Rates (Needs Updated With New Information)
7. Early Registration Discounts
8. Gives Credits
9. Enter and Update Tuition Rates
10. Enter and Update Fee Rates and Type (Needs Updated With New Information)
11. documentation
12. Trello management

Dicheng:

1. 3.5 Tie function together
2. 3.5 Class Search Pages
3. 3.5 Updated Role sheet

4. 3.5 Research Linux Boxes
5. 3.5 Tie Functions Together Into a Single Prototype
6. Enter Payments From Multiple Sources For One or More Students
7. Look at The Billing History
8. Enter a Full Payment For One or More Students
9. Enter Payments From Multiple Sources For One or More Students
10. Generate an invoice for the amount due (Needs Updated)
11. Look at a The Current Amount a Student/Family Owes
12. View Teaching History
13. Auto and Manual discount giving
14. Set semester

Together:

1. Updated database construction
2. GUI page breakdown based on user stories and product backlog
3. Coded some functionality

## 5 Sprint Report #5

Sprint Report #5

### 5.1 Team Members:

Marcus Berger

Dicheng Wu

**Sponsor:**

Jeff McGough

### 5.2 Prototype Progress

Sprint five was dedicated to finishing the last of the functions and clean up of the project. First the client modified some requirements for the team. After this the user stories remaining from sprint four were tackled by the group to produce the remaining payroll and billing interface. This included logging hours, and payments. After this was done the team started on bug fixes and quality of life updates such as new buttons and removal functions. The team is currently working on finishing up the student registration. If progress continues at this rate the prototype should be completed functionally for our teams iteration however without some work the prototype may not reach some of the teams desired standard.

### 5.3 Project deliverables of Sprint 5:

Sprint 5 deliverables were the remaining functionality for the redefined requirements from the client for the project. If the sprint was successful the main project will be done or very close to done functionally for this iteration at the end of the sprint.

### **5.3.a Sprint 5 Backlog**

1. Admin and teacher update crossover
2. Ignoring case in address forms
3. Add rejected students to approved/ reject and provide current status
4. Multiple pay rates
5. Admin list function
6. Enter staff hours
7. Sprint 4 rollover (remaining payroll functions)
8. Add and remove class location
9. Removal Functions (Admin, Class, Student, Teacher)
10. Refunds and check boxes for fees and tuition
11. Quality of life updates to some functions
12. Bug Fixes
13. Student registration
14. Employee wages
15. Begin User Guide for documentation

### **5.4 Sprint 5**

During sprint five the team tackled the remaining functionality, bugs, and quality updates in an attempt to finish the base project. Removal functions to clean out the database were added, and some client requested quality of life updates were added for this project version. Bugs and glitches were patched up in many functions. Another job of this sprint was to finish the rollover from sprint four since some payroll functions still needed work. Lastly the team tackled student registration since the student interface was dropped from the project requirements in sprint 4. The registration is still in progress at this time, as the team failed to complete this functionality in time for the end of the sprint. However it was completed during sprint 6.

### **5.5 Prorated Refunds**

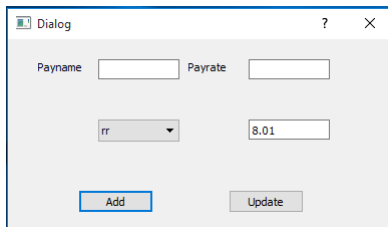
The prorated refunds code will keep track of how much a class costed a student. If the student drops a class it will refund the remaining worth of the class to the students school credit field. This file has now been completed.

### **5.6 Enter Staff Hours**

This page needed to undergo changes after meeting with client. The new version will allow a teacher to log their in class hours, office hours, trip hours, etc. These hours will then be paid out at their different rates respectfully. The hours will be logged as single numbers not by days or weeks. This function is now completed

## 5.7 Enter Teacher Wages

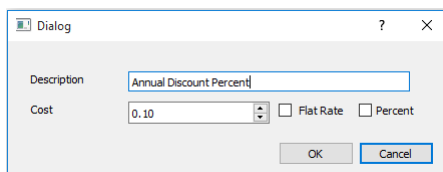
This was handled in the database in previous sprints assuming one pay rate. However in talks with the client the academy pays different pay rates based on what part of academy work they are doing. So the page will need to be updated once the list of possible pay rates have been provided by the client. This pages and its updates are still in progress.



A screenshot of a 'Dialog' window titled 'Dialog' with a question mark icon and a close button. The window contains two text input fields labeled 'Payname' and 'Payrate'. Below 'Payname' is a dropdown menu showing 'rr'. To the right of the dropdown is a text input field containing '8.01'. At the bottom of the dialog are two buttons: 'Add' and 'Update'.

## 5.8 Enter Tuition Rates and Fees

Updated these pages to provide quality of life updates to the function requested by the client.



A screenshot of a 'Dialog' window titled 'Dialog' with a question mark icon and a close button. The window contains a 'Description' field with the text 'Annual Discount Percent'. Below it is a 'Cost' field with a value of '0.10' and a small spinner icon. To the right of the 'Cost' field are two checkboxes: 'Flat Rate' and 'Percent'. At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

## 5.9 Bug Fixes

During the course of testing and the client meeting some bugs or inconsistencies were discovered within the project. The bugs found and fixed are:

1. Student search was not showing all the students. Status: Fixed
2. Search edit need to be turned off and advance search modifications. Status: Fixed
3. Search, role, and schedules need to have dynamic not static times. Status: Fixed
4. Allow for time changes on schedule. Status: Fixed
5. Some of the back buttons in the teacher interface did not work correctly. Status: Fixed
6. Forms seem to be bugged at times, however can not recreate bug. Status: In testing possibly fixed.

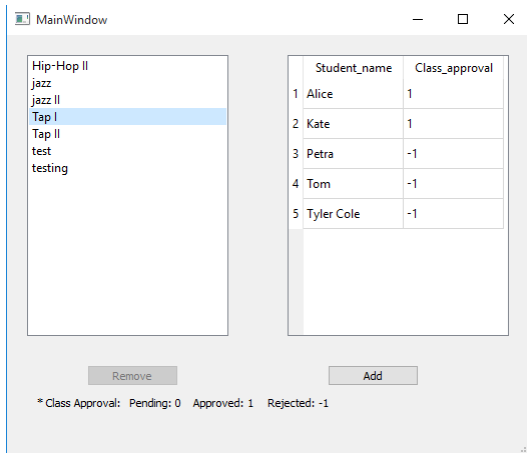
## 5.10 Updates Crossover

One of the updates that needed to be completed this sprint was the need for admin and teacher updates to crossover if something was changed. This means that if someone is an admin and they update there phone number through the "update my information form" then the phone number will be updated in both the teacher table and the admin table. This way the system will not have conflicting information in different places.

## 5.11 Approved/Rejected Student Updates

A update requested by the client was the ability to see rejected students in the approved/rejected pages just in case the academy changed its mind about a student. The page underwent slight modifications to the way information was displayed to make this change efficiently



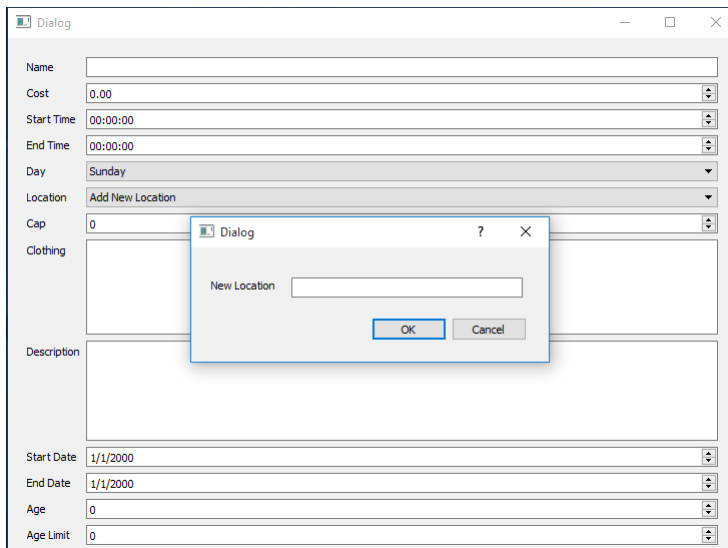


## 5.12 Admin List

In creating some of the updates it became clear that a specific admin list was needed to see who had admin access to the system. This was done through a simple list view of the names that can then be selected to display more detailed information.

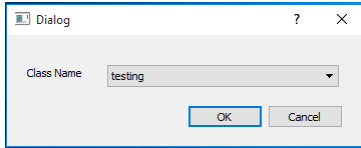
## 5.13 Add/Remove Locations

Another requested feature was the ability to add and remove locations for classes in the database. This feature is necessary because the academy sometimes teaches classes in different places based on need. The team accomplished this using a simple dialog that connects to the location table in the database. When the user updates or adds a new class the location combo box provides a list of locations in the database and an add location option.



## 5.14 Removal Functions

These are simple functions that pull up a dialog box where the user can select a name and remove all the information related to that user from the system. The removal functions that exist in the system are: teacher, admin, student, and class. Teacher will remove that persons information from the system. Admin removes that persons admin rights and then ask if the whole user should be removed.



### 5.15 Sprint 5 Issues

Some issues that were encountered during this sprint were:

1. Spring semester ramp up - This was by far the biggest issue this sprint and the main cause of the failures this sprint. Other classes up to the point of blocking work on many projects. Staggered due dates meant that when the team wanted to work, there was always another projects that needed to be completed in a short amount of time. By the end much of our time had been taken and the sprint began to fall behind.
2. Requirement redefining - Due to drop the student interface the functionally needs to be added to the GUI which created some extra work for the team.
3. Failure to complete backlog - The team did not finish all of the assign backlog this sprint.

### 5.16 Client Interactions

Client interactions came in the form of required reviews and a demo of the prototype at the end of the sprint. Topics included:

1. Progress reports on where we are in the project
2. Asking questions to refine functionality, understand academy needs, and update requirements.
3. Discuss the future of the project and idea on how to execute them.
4. Prototype the current version of the prototype.

### 5.17 Group Meeting

The group has a standard meeting time of TTH 10:00 - 12:00 and once during the weekend and at home as needed.

Meeting can continue pass these times if needed, and other times during the weekend. However extra weekend times fluctuate and do not remain constant from week to week.

### 5.18 Work Distribution

Marcus:

1. Bug fixes
2. Removal functions
3. Admin and teacher update crossover
4. Ignore Case on address forms
5. Add rejected students to appoved/ reject and provide current status
6. Add/remove class location
7. Prorated refunds
8. Quality of life updates

9. Sprint report
10. Start User Guide
11. Trello management

Dicheng:

1. Bug fixes
2. Pay rates
3. Admin list
4. Staff hours
5. Student registration (in progress)
6. Wage calculations (in progress)

Together:

1. Updated database construction
2. Update prototype structure
3. Coded some functionality



# C

---

## Industrial Experience and Resumes

---

### 1 Resumes

## **Marcus J. Berger**

4534 Bozeman Cir.

Rapid City, SD 57703

Cell: 605-430-2940

[marcus.berger@mines.sdsmt.edu](mailto:marcus.berger@mines.sdsmt.edu)

### **Employment**

**Intern,** SDSM&T University Relations, Rapid City, SD 9/2013 to 9/2015

- Managed and created website content utilizing HTML and style sheets
- Experience with Content Management Systems (Estrada, Ektron)
- Marketing Exposure through SDSM&T PR campaigns and programs
- Helped develop a project that won a Black Hills American Advertising Award

**Computer Landscape Designer,** Turf and Erosion Solutions, Rapid City, SD 5/2013 to 8/2013

- Created and set up landscapes on computer for customers
- Created mock-ups to help customers visualize projects such as large company buildings

### **Education**

**Bachelor of Science in Computer Science** Anticipated: 5/2016

South Dakota School of Mines and Technology, Rapid City, SD

GPA 2.86

- **Courses including but not limited to:** Assembly Language, Computer Graphics, Computer Networking, Database, and Data Mining
- **Current Major Courses:** Artificial Intelligence, Graphical User Interfaces, Operating Systems, and Senior Design
- **Projects include:** ANN, Database projects including PHP and SQL, Data Manipulation, Image Processors (C++ and ARM Assembly Language), Semi-Supervised Learning, Socket Programming, and a Solar System Model in OpenGL
- **Senior Design Project:** Developing an open source program, database and GUI to handle student information and other business information for the Academy of Dance Arts in Rapid City.

### **Qualifications**

#### Professional Skills:

Experience with Agile development and Github  
Knowledge of Waterfall development  
Task Driven Development  
Working in Teams  
Oral and Written Communication

#### Software Skills:

Fluent in C++  
Fluent in Python and PyQt  
Experience with HTML, PHP and SQL  
Lisp  
Webpage Design (CMS, Google Analytics, etc.)  
Some Java and Swing

**References – Available Upon Request**

# Dicheng Wu

729 E Anamosa St. #101  
Rapid City, SD 57701

(605) 389-6729  
dicheng.wu@mines.sdsmt.edu

## EDUCATION

**South Dakota School of Mines and Technology** - Rapid City, SD

- Computer Science B.S.
- Expected Graduation Date: May 2016

**Overall GPA: 3.36**

**Major GPA: 3.67**

### Related Courses

- Data Structures
- Analysis of Algorithms
- Software Engineering
- Database Management
- Assembly Language
- Digital Image Processing
- Programming Language
- Natural Computing
- Senior Design I
- Introduction to Robotics
- Data Mining & Visualization
- Networking/Data Communication

## TECHNICAL SKILLS

- **Proficient:** C/C++, Python, JAVA
- **Experienced:** Common Lisp, SQL, ARM Assembly
- **Libraries:** CUDA, PyQt, NumPy, SciPy, Matplotlib
- **Operating Systems:** Windows, Linux
- **Database Systems:** MySQL, MSSQL
- **Environment:** Visual Studios, NetBeans, Qt Creator, LaTeX, Github, Agile Development

## PROJECTS

- **Evolving Neural Networks To Play Tic-Tac-Toe:** The program is written in C++. The main functionalities of this program are to train a neural network by using genetic algorithm and to play tic-tac-toe with humans. I also rewrite the neural network with applying back-propagation and CUDA afterwards.
- **Dance Soft:** This is a senior design project. We are making an open-source software which will run on local Dance Academy for their students and classes' management. The software is written in Python and PyQt and built upon MySQL database. The development process strictly conforms to "agile development methodology".

## PROGRAMMING CONTEST

- Placed 47<sup>th</sup> (out of 290 teams) at 2014 ACM-ICPC North Central North America Regional.

## EXPERIENCE

- **South Dakota School of Mines and Technology Dining Service** - Jan 2015 – May 2015  
Cleaning tables and washing dishes.

## **2 Industrial Experience Reports**

### **2.1 Marcus Berger**

Marcus has had an internship with the web development branch of the University Relations department of South Dakota School of Mines and Technology. While not considered a direct computer science internship Marcus was able to pick up various web programming principles such as HTML, Formatted CSS, Content Management Systems, and other web technology such as analytic information. Marcus spent two years working for the University Relations from the Fall 2013 to Fall 2015.

Outside of the intern with the South Dakota School of Mines, Marcus has had no other internships or industrial experience while attending School of Mines. Once graduated, Marcus has excepted a job with the software development company Computers Unlimited located in Billing Montana. Marcus will officially start at Computers Unlimited on June 6th 2016.

### **2.2 Dicheng Wu**

As of the DanceSoft project turn in date Dicheng Wu has had no internships or industrial experience of note. However once the project is submitted and the team has graduated from SDSMT, Dicheng currently plans to move to Fargo, North Dakota to work for the branch of Microsoft located there.



## D

---

### Acknowledgment

---

The team would like to acknowledge the following people and groups from their input and assistance on various aspects of the DanceSoft project:

1. Brian Butterfield - For assistance in senior design class and project input
2. Dr. Mengyu Qiao - For assistance with database construction, and SQL references
3. Roger Schrader - For assistance in storing the database and computer access
4. Computers Unlimited Senior Design Group - For providing feedback and input on the project and the senior design class.
5. ARM Cluster Research Team - For providing feedback and input on the project and the senior design class.
6. Fellow Members of the Senior Design Class - For input, recommendations, and assistance during project development



**E**

---

**Supporting Materials**

---

