ICT2101/2201
Introduction to Software Engineering

Milestone 2: Software Design Specifications & Management

for

Music School

Prepared by

Team's Name: AI

Ang Wee Yi, Alex	**2102852**
Iphigene Peh Yureng	**2102741**
Iphigena Peh Yuxi	**2102744**

Lab Group:	P2-9
Github handle:	ICT2201-P2-9
Date:	29/10/2022

Table of Contents

1 Introduction

In the music school, work allocations are manually assigned monthly to teachers by the managers. Hence, to facilitate better workload allocation, the team has built a web-based workload management system to show an overview of the company's manpower strength at any time and informative availability of employees and engagement. The system will provide an interactive and informative way for both employees and managers to visualise and allocate their workload optimally which will in turn improve work-life balance and make the workplace environment more conducive and attractive. The following subsections are organised as such: Subsection 1.1 describes the scope of the product. Subsection 1.2 shows the related background literature. Subsection 1.3 discusses the intended audience and the overview of the document. Finally, subsection 1.4 included references and acknowledgements.

1.1 Product Scope

The scope of the product is to create a web-based application that will be utilised by managers, staff and IT administrators for workload allocation in the Music School.

The employees of Music School should be able to see their job assignments and working hours engaged/assigned. They will also be able to indicate their availability up to one month earlier and indicate if they cannot fulfil any assigned jobs. This product would greatly benefit the employees as they will now have a clear overview of their work schedules, allowing them to plan both days off and working days conveniently.

In addition, the managers should be able to visualise the manpower availability up to one month earlier, as well as job assignments and allocate jobs. This product would benefit managers as they are able to view regular updates on staff availability and engagement more effectively, allowing them to follow up and plan an optimised workload schedule.

Furthermore, the IT administrators will oversee staff and managers accounts by creating, editing or removing them. This product would be useful for the IT administrators as they could manage accounts more efficiently as all accounts are in a centralised system.

1.2 Intended Audience and Document Overview

The intended audience of this document are the client and the professor. In this project, the client refers to an entity that has engaged the team to design and build a custom software according to their requirements and specifications. The professor would provide relevant and essential feedback throughout the project and documentation process for the team and ensure this document meets the stated requirements and specification.

The recommended sequence for reading the document is:

1. Section 1 offers a good overview and understanding of the entire project.
2. Section 6, 7 and 8. These sections provide useful information regarding individual use cases descriptions and a data dictionary that consists of the different terminology used in this document to prevent any misunderstanding.
3. Section 2 illustrates the specific requirements of the product which comprises the requirement elicitation methods, product overview, product functionality and requirements, and updated use case model from milestone 1.
4. Section 3 details the software design of the product. The software design consists of different diagrams such as the architecture diagram, C4 model, entity class modeling, class diagram, sequence diagram and communication diagram.
5. Section 4 would be most pertinent to the client as it encapsulates the project plan which includes the following subsections: software estimation and project management plan. These sections would give the client a rough estimation of the project cost, resources and time required before the release of the product.
6. Section 5 consists of individual team reflection regarding this project.

In general, all of the sections are pertinent to the professor for grading and reviewing the team effort and work done.

1.3 References and Acknowledgments

[1] D. E. Guest, “Perspectives on the study of work-life balance - david E. guest, 2002,” Sage Journals, Jun-2002. [Online]. Available: https://journals.sagepub.com/doi/10.1177/0539018402041002005.

[2] I. Lupu and M. Ruiz-Castro, “Work-life balance is a cycle, not an achievement,” Harvard Business Review, 29-Jan-2021. [Online]. Available: https://hbr.org/2021/01/work-life-balance-is-a-cycle-not-an-achievement.

[3] R. Knight, “Make sure your team's workload is divided fairly,” Harvard Business Review, 03-May-2017. [Online]. Available: https://hbr.org/2016/11/make-sure-your-teams-workload-is-divided-fairly.

[4] H. Inegbedion, E. Inegbedion, A. Peter, and L. Harry, “Perception of workload balance and employee job satisfaction in work organisations,” Heliyon, 09-Jan-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844020300050.

[5] B. Roseke and A. B. Roseke, “Learn,” ProjectEngineer, 19-Feb-2016. [Online]. Available: https://www.projectengineer.net/estimating-task-durations/.

2 Specific Requirements

This section details the new and changed requirements of the product which includes user interface requirements, functional requirements, updated use case model, non functional requirements, performance requirements, and safety and security requirements.

2.1 Requirements Elicitation Method

The process of requirement discovery is crucial to focus on and understand what the client needs for the product. Hence, the team has utilized several requirements elicitation methods to derive the specific requirements.

The first requirement elicitation method utilized was via Documents. The team was given a document with an overview of the project descriptions which includes most of the functional system requirements and user requirements. The document provides the team with an understanding of the problem domain and business functionality which aids in the initial process of requirement discovery and provides the essential materials for the second requirement elicitation method.

The second requirement elicitation method utilized was conducting interviews. A client meeting was set during the second week of the assignment to gather user interface requirements, functional and nonfunctional requirements, performance requirements, and safety and security requirements. This elicitation method is beneficial for the team as it provides the team with the opportunity to clarify requirements and doubts on the documentation provided by the client through face-to-face interaction.

The third requirement elicitation method utilized was prototyping. This elicitation method allows the team to first create a prototype based on the known requirements and specification for the client. The prototype is built quickly and showcased to the client during the client meeting to gather feedback and opinions regarding the mock up product. The prototype gives a clearer picture of what the client needs, and at the same time, provides assurance to the client as well as the team on the direction of the project.

2.2 Product Overview

The product is a web-based application that replaces paper records and manual processes. The web-based application will be built with ExpressJS, which is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. The web server will be hosted in application instances run within Docker containers on Compute Engine virtual machines (VM).

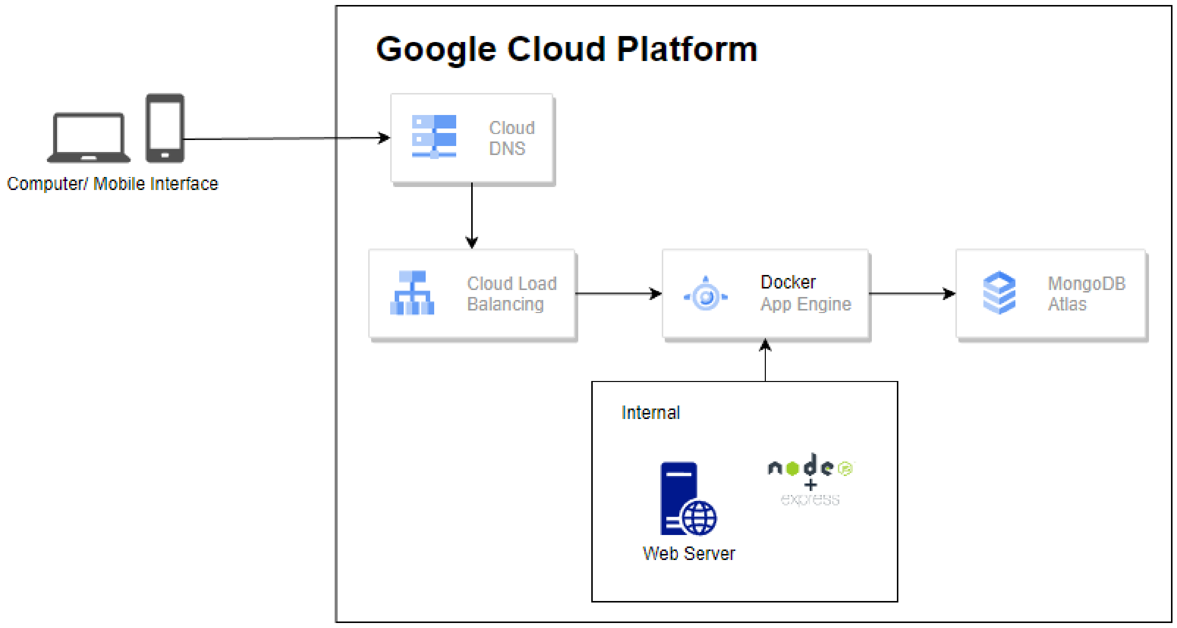


Figure 1: System Architecture Diagram

Figure 1 presents an overall view of the system architecture. The web-based workload management system provides functionalities that are stated in Section 2.2 Product Functionality. It has interfaces to external systems, such as the Cloud Database System. The web-based workload management system stores all its data in the Cloud Database System and it is therefore essential to maintain a connection to it. The team would be using Google Cloud as its cloud database as it provides AES256 encryption for data stored in the database. This would ensure that the stored data is more secure and safe for the company.

2.3 User Interface Requirements

The user interface requirements (UR) includes the following:

UR1: The system shall use a colour theme of red, blue, orange, green, purple, grey, black and white.
UR2: The system shall use a background page that shows different types of instruments.
UR3: The system shall support mobile applications and web browsers.
UR4: The system shall use black fonts for wordings and title.
UR5: The system shall include drop-down menus and buttons to assist the selection of options.

2.4 Functional Requirements

This section consists of all the functional requirements (FR) that capture the intended behaviour of the system.

FR1: The system shall display job assignments in a weekly schedule in the landing page to the staff.
FR2: The system shall display overall workload in the landing page to the staff
FR3: The system shall allow staff to submit their available dates with the respective time slots up to 5 weeks in advance.
FR4: The system shall allow staff to reject jobs allocated to them from Thursday to Friday each week to send a prompt to notify the manager.
FR5: The system shall allow staff who are qualified to teach more than one instrument to indicate their preference on which lesson they prefer to teach.
FR6: The system shall display information about the overall workload allocated for different instruments in the landing page to the manager.
FR7: The system shall highlight the top three staff with highest workload in the landing page to the manager
FR8: The system shall allow managers to allocate jobs to staff in 30 minute sessions while checking if the staff exceeded four consecutive hours of work without rest and an available studio is allocated to the job.
FR9: The system shall allow managers to view staff's workload such as jobs assigned, workload allocated in hours and availability.
FR10: The system shall allow IT administrators to manage the user accounts.

2.5 Updated Use Case Model

The use case model diagram in Figure 2 is an updated version of the model from milestone one. The changes include the addition of extension points in the diagram.

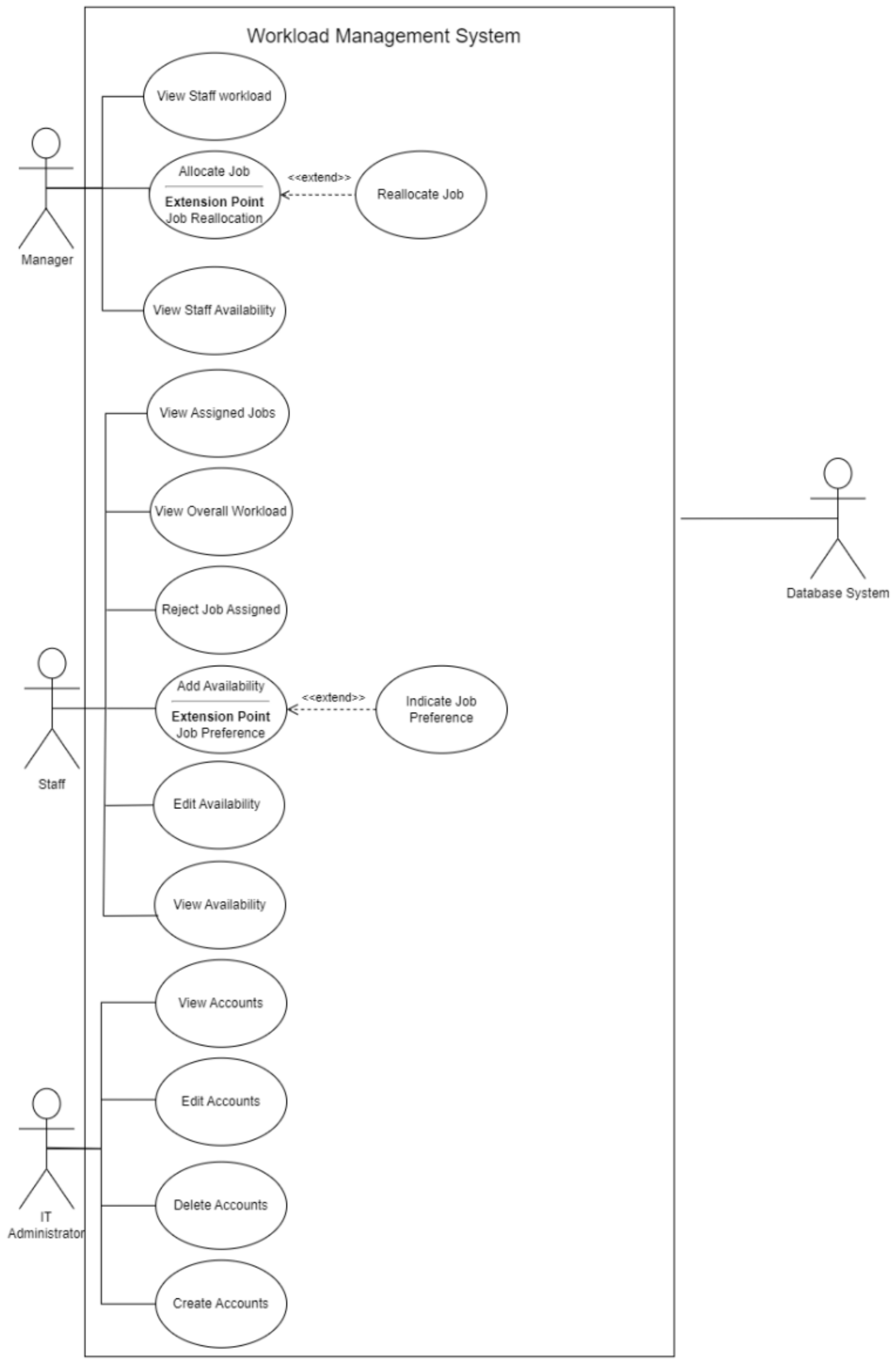


Figure 2: Use Case Model

2.6 Non-functional Requirements

The non-functional requirements (NFR) include the following:

NFR1: The system shall present content in English.
NFR2: The system shall be designed with responsive design and mobile-friendly for all types of mobile devices such as mobile phones and tablets.
NFR3: The users shall be able to use all the system functions after one hour of training. After this training, the average number of errors made by experienced users shall not exceed two per hour.

2.6.1 Performance Requirements

The performance requirements include the following:

NFR4: The server will recover after experiencing a crash or denial of service within 24 hours

2.6.2 Safety and Security Requirements

The safety and security requirements include the following:

NFR5: Confidential data stored in the database shall be encrypted with AES-256.
NFR6: All approving roles must use a different factor of authentication from the user authentications (on top of 2FA) for approval processes.
NFR7: The system shall have multi-factor authentication (MFA).
NFR8: The system shall enforce strong password policy where passwords have to contain:

- At least 1 special character/ symbol
- At least 8 characters (Mixture of uppercase and lowercase)
- At least 1 number
- Change password every 90 days

NFR9: The data stored in the database shall be backed up to prevent data loss.
NFR9: The website shall have a Secure Sockets Layer (SSL) certificate to secure communications between the web server and client.

3 Software Design

The team's software philosophy is to abide by the S.O.L.I.D Principles in order to achieve a good software design.

The Single Responsibility Principle ensures that classes have only one main responsibility. Therefore, classes should be separated into different and isolated layers such as the presentation layer, data layer, and business logic layer. The Open-Closed Principle states that software entities can be open for extension without modifying their source code. The Liskov Substitution Principle models good inheritance, which promotes reusability of codes during the coding phase. The Interface Segregation Principle is closely associated with the Single Responsibility and Liskov Substitution Principles as it ensures specific interfaces are created for the purpose of a particular class and implements the mentioned interfaces only if necessary. The Dependency Inversion Principle helps us to couple software modules loosely, which ensures that high-level modules with complex logic should be easily reusable and unaffected by low-level modules.

3.1 Architectural Design

The software architectural design chosen for this project is a 3-tier architecture. With considerations in mind that our team is small, having reduced dependency between the different tiers allows us to concurrently work on different parts of the project as well as scalability in the future without affecting other tiers. In addition, it is easy to learn and implement for the short timeline that will increase the development speed. In addition to this, the technology stack that we have chosen for this project is MongoDB, ExpressJS, NodeJS, and the AngularJS (MEAN) stack. In this technology stack, TypeScript is the core language being used in AngularJS, ExpressJS, and NodeJS, which allows us to learn and apply to different parts of the project.

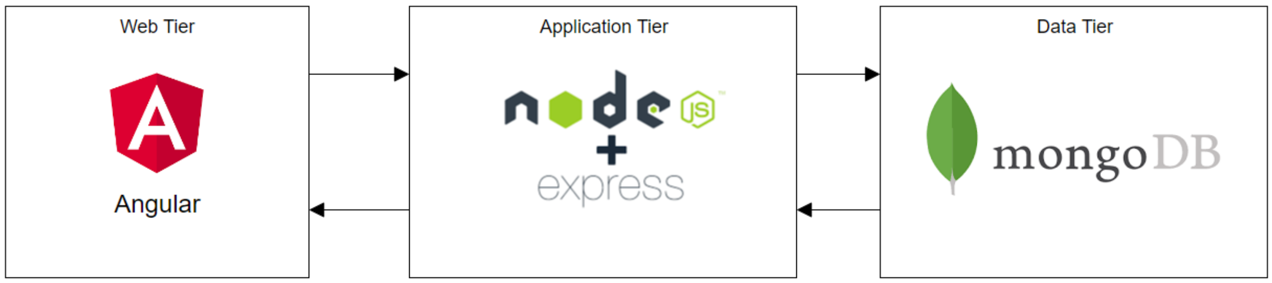


Figure 3: 3-tier Architecture Diagram

Figure 3 illustrates a 3-tier architecture where the angular framework serves as the presentation tier to handle the user interface and communication with the application tier. In the application tier, ExpressJS, which is layered upon NodeJS, is used as the backend framework to design APIs to communicate with the presentation tier and request data from MongoDB, which serves as the data tier.

This section illustrates the software architectural design with the aid of component diagrams to present an overview of how certain parts of the project come together as one.

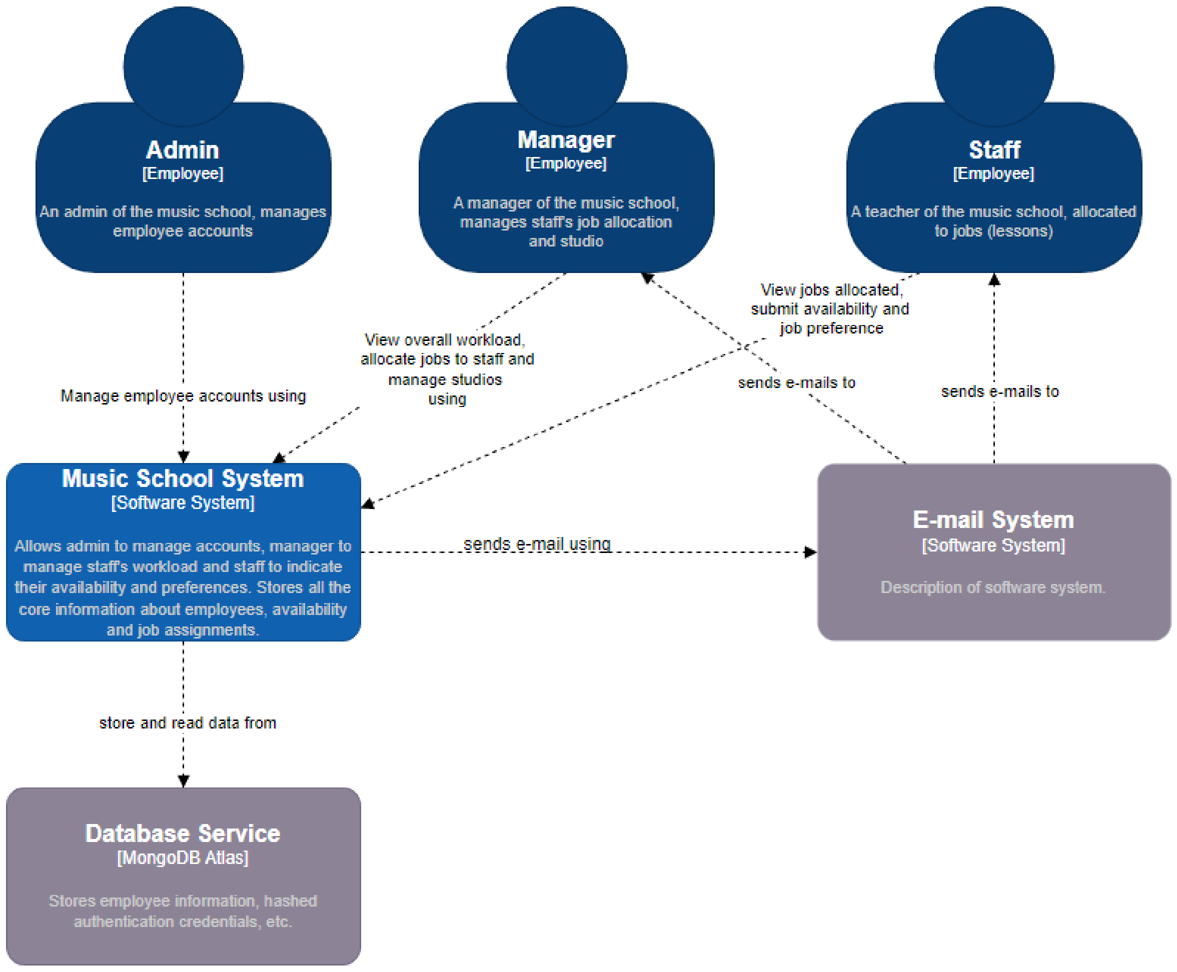
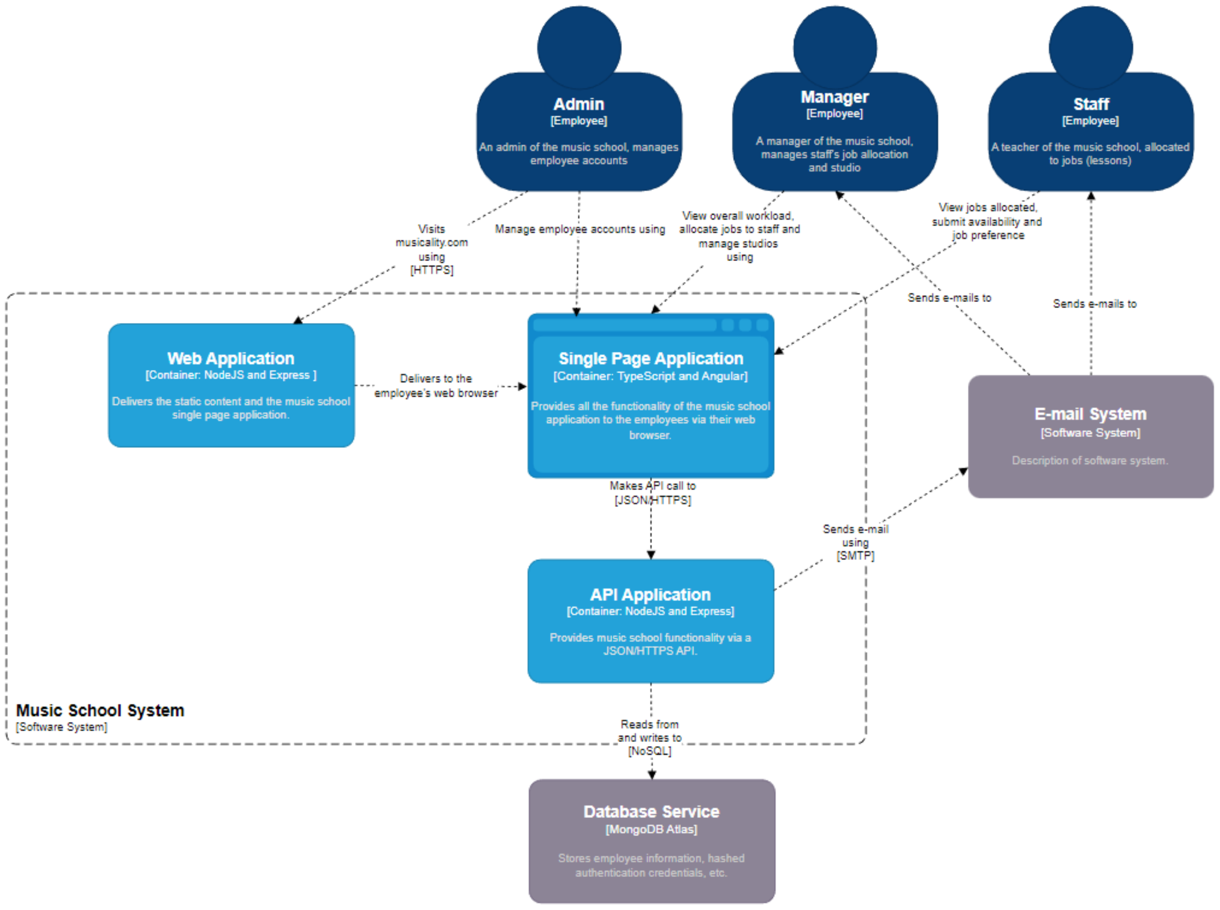

Admin
[Employee]
An admin of the music school, manages employee accounts
Manager
[Employee]
A manager of the music school, manages staff's job allocation and studio
Staff
[Employee]
A teacher of the music school, allocated to jobs (lessons)
Manage employee accounts using
View overall workload, allocate jobs to staff and manage studios using
View jobs allocated, submit availability and job preference
sends e-mails to
sends e-mails to
Music School System
[Software System]
Allows admin to manage accounts, manager to manage staff's workload and staff to indicate their availability and preferences. Stores all the core information about employees, availability and job assignments.
sends e-mail using
E-mail System
[Software System]
Description of software system.
store and read data from
Database Service
[MongoDB Atlas]
Stores employee information, hashed authentication credentials, etc.

Figure 4: System Context Diagram

The system context diagram, as shown in Figure 4, illustrates the system scope and interaction with different users such as Admin, Manager, and Staff to perform different actions. Other actors include an external database and an external email system. This diagram provides a high-level abstraction suitable for both technical and non-technical people.

The container diagram as shown in Figure 5 describes the software system, showing the interactions between the Angular single-page application, Express API, and MongoDB database that were chosen for this software system. This diagram provides a high-level technology-focused diagram suitable for developers and software architecture.



Admin
[Employee]
An admin of the music school, manages employee accounts
Manager
[Employee]
A manager of the music school, manages staff's job allocation and studio
Staff
[Employee]
A teacher of the music school, allocated to jobs (lessons)
Visits musicality.com using [HTTPS]
Manage employee accounts using
View overall workload, allocate jobs to staff and manage studios using
View jobs allocated, submit availability and job preference
Sends e-mails to
Sends e-mails to
Web Application
[Container: NodeJS and Express]
Delivers the static content and the music school single page application.
Delivers to the employee's web browser
Single Page Application
[Container: TypeScript and Angular]
Provides all the functionality of the music school application to the employees via their web browser.
E-mail System
[Software System]
Description of software system.
Makes API call to [JSON/HTTPS]
Sends e-mail using [SMTP]
API Application
[Container: NodeJS and Express]
Provides music school functionality via a JSON/HTTPS API.
Music School System
[Software System]
Reads from and writes to [NoSQL]
Database Service
[MongoDB Atlas]
Stores employee information, hashed authentication credentials, etc.

Figure 5: Container Diagram

The layer 3 component diagram as shown in Figure 6 describes the components that exist within the backend API and how they interact with each other in detail. This provides clarity for the developers during implementation to structure their code to ensure code reusability as well as to concurrently work on different components at the same time. This diagram provides the responsibilities and implementation details of the different components suitable for developers and software architects.

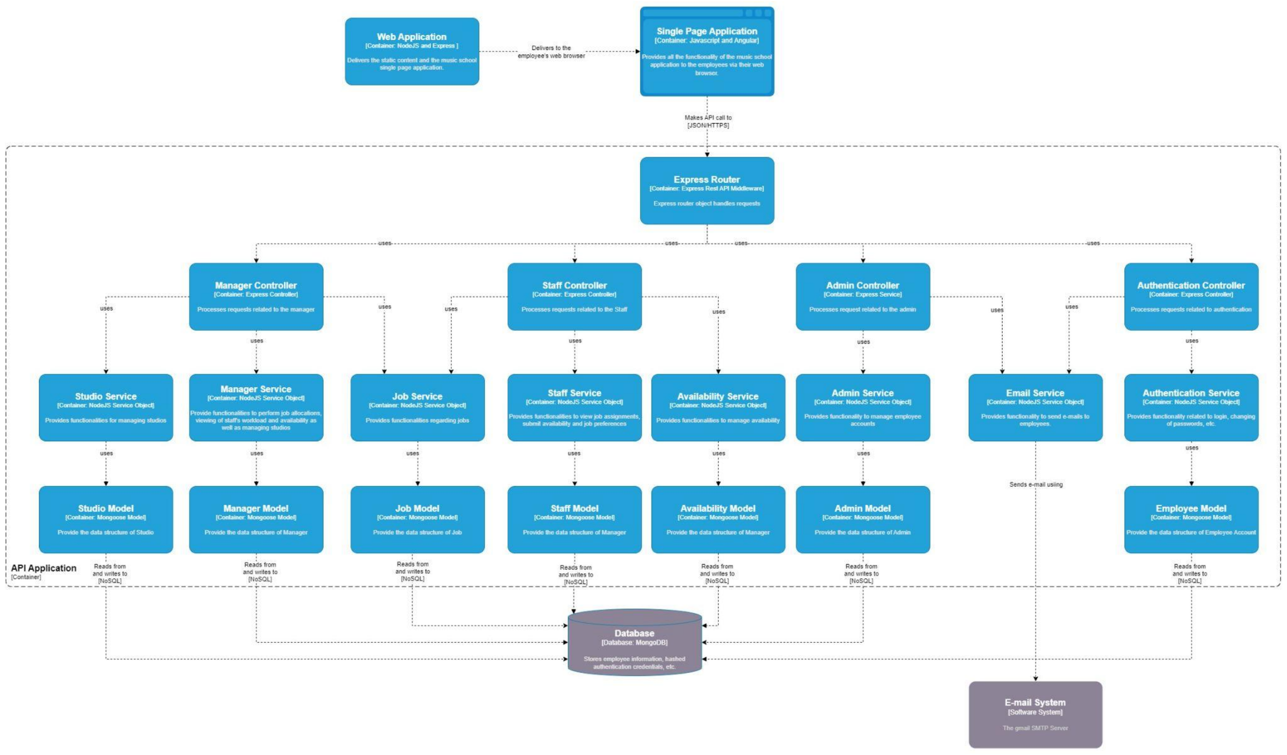

Web Application
[Container: NodeJS and Express]
Delivers the static content and the music school single page application.
Delivers to the employee's web browser
Single Page Application
[Container: Javascript and Angular]
Provides all the functionality of the music school application to the employees via their web browser.
Makes API call to [JSON/HTTPS]
Express Router
[Container: Express Rest API Middleware]
Express router object handles requests
uses
Manager Controller
[Container: Express Controller]
Processes requests related to the manager
Staff Controller
[Container: Express Controller]
Processes requests related to the Staff
Admin Controller
[Container: Express Service]
Processes request related to the admin
Authentication Controller
[Container: Express Controller]
Processes requests related to authentication
Studio Service
[Container: NodeJS Service Object]
Provides functionalities for managing studios
Manager Service
[Container: NodeJS Service Object]
Provide functionalities to perform job allocations, viewing of staff's workload and availability as well as managing studios
Job Service
[Container: NodeJS Service Object]
Provides functionalities regarding jobs
Staff Service
[Container: NodeJS Service Object]
Provides functionalities to view job assignments, submit availability and job preferences
Availability Service
[Container: NodeJS Service Object]
Provides functionalities to manage availability
Admin Service
[Container: NodeJS Service Object]
Provides functionality to manage employee accounts
Email Service
[Container: NodeJS Service Object]
Provides functionality to send e-mails to employees.
Authentication Service
[Container: NodeJS Service Object]
Provides functionality related to login, changing of passwords, etc.
Sends e-mail usiing
Studio Model
[Container: Mongoose Model]
Provide the data structure of Studio
Manager Model
[Container: Mongoose Model]
Provide the data structure of Manager
Job Model
[Container: Mongoose Model]
Provide the data structure of Job
Staff Model
[Container: Mongoose Model]
Provide the data structure of Manager
Availability Model
[Container: Mongoose Model]
Provide the data structure of Manager
Admin Model
[Container: Mongoose Model]
Provide the data structure of Admin
Employee Model
[Container: Mongoose Model]
Provide the data structure of Employee Account
API Application
[Container]
Reads from and writes to [NoSQL]
Database
[Database: MongoDB]
Stores employee information, hashed authentication credentials, etc.
E-mail System
[Software System]
The gmail SMTP Server

Figure 6: Layer 3 Component Diagram

The layer 4 component diagram shown in Figure 7 describes the 3 tier architecture in detail, from the presentation tier running on AngularJS that exists in the user's web browser to the backend ExpressJS server that exists on the cloud and communicates to an external MongoDB database that exists on the same cloud. The details within this component diagram will be further elaborated using the class diagram.

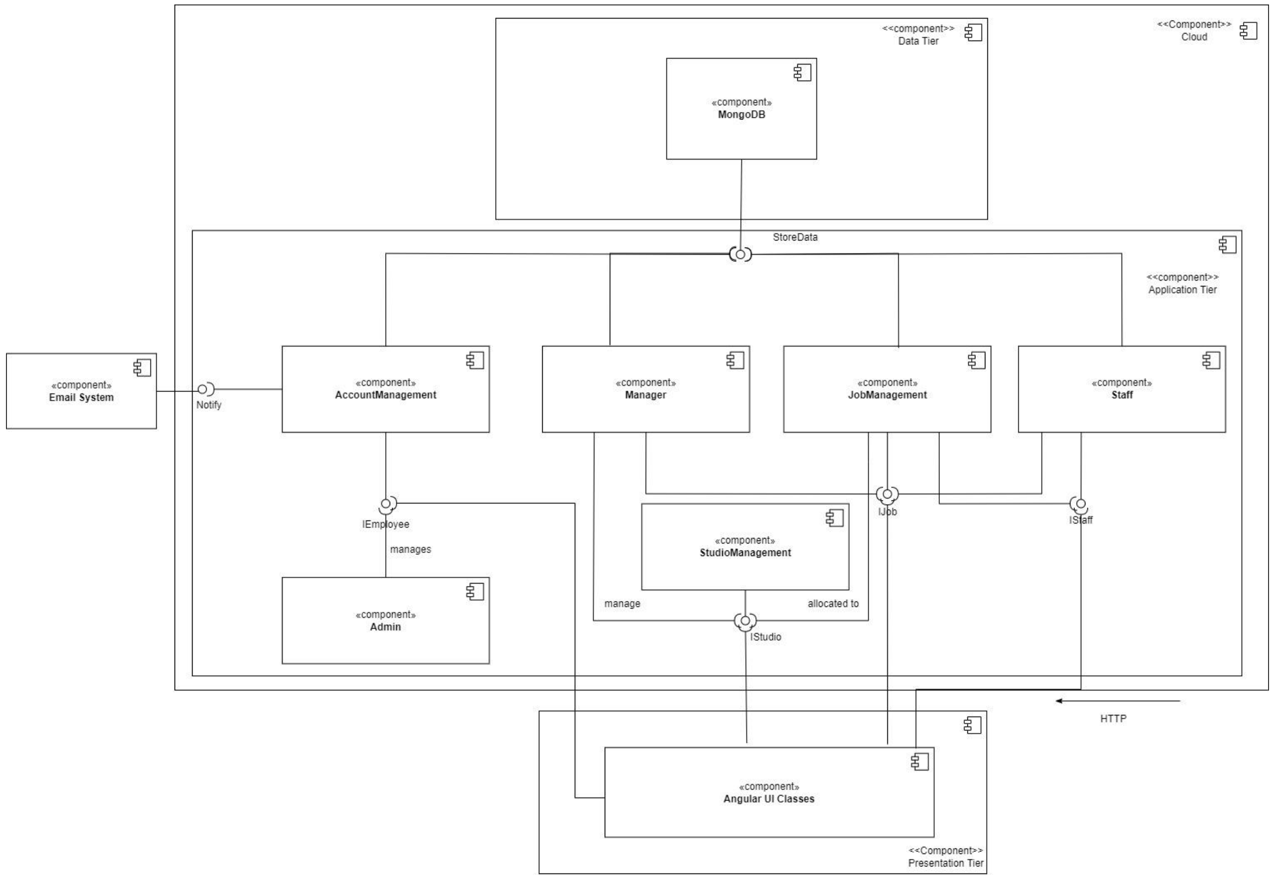


Figure 7: Layer 4 Component Diagram

Component	Classes
AccountManagement	Employee, EmployeeController, EmployeeFactory, IEmployee
Manager	Manager, ManagerController
JobManagement	Job, JobController, IJob
Staff	Staff, StaffController, Preference and Availability, IStaff
Admin	Admin, AdminController
StudioManagement	Studio, StudioController, IStudio
Angular UI Classes	AccountManagementUI, AllocateJobUI, ManagerDashboardUI, AddAvailabilityUI, StaffDashboardUI, AddPreferenceUI, StudioUI and LoginUI

Table 1: Classes of Component

3.2 Detailed Design

This section illustrates the detailed design of the software with the aid of software classes, class diagrams, and sequence diagrams to provide an overview of the project and the design flow between the objects.

In the class diagram, “Accessor Methods()” represents the getter and setter for each respective attribute in the class.

Software Classes

Employee Entity Class
The Employee class stores the base properties that every user of this Music School System will inherit from.

EmployeeController Control Class
The EmployeeController class uses the Employee class to do authentication for login and uses the EmployeeFactory class to create Manager and Staff. This class also provides an interface for the AdminController control class for the admin to manage the employee accounts.

Staff Entity Class
The Staff class inherits from the Employee class and has an additional attribute for the instrument that the Staff is qualified to teach. The staff class also contains other classes such as Availability and Preference. In addition, the Staff have different behaviors from the Manager and Admin. Thus, the Staff class is used by a separate control class, StaffController, for indicating availability, preference, and viewing workload.

StaffController Control Class
The StaffController class uses the Staff, Availability and Preference entity classes to provide the processing and execution of the use cases required by the staff. This control class also requires an interface from the JobController control class to reject jobs and provides an interface for the JobController control class to check on the staff’s availability.

Admin Entity Class
The Admin class inherits from the Employee class and is used by a separate control class, AdminController for managing the employee accounts of both Manager and Staff.

AdminController Control Class
The AdminController class uses the Admin class and requires an interface from EmployeeController to manage the employee accounts.

Manager Entity Class
The Manager class inherits from the Employee Class and is used by a separate control class, ManagerController, for viewing staff workload, allocating jobs to the staff.

ManagerController Control Class
The ManagerController class uses the Manager entity class and requires an interface from the JobController control class to allocate jobs to the staff. This class also requires another interface from StudioController for Manager to manage the studios.

Job Entity Class
The Job class stores the properties of a lesson, such as type of lesson, date, and time of the lesson. This job class has a composition relationship with the staff and only belongs to a single staff at a time. This class is also used by the JobController control class for creating jobs and rejecting jobs.

JobController Control Class
The JobController control class uses the job entity and requires an interface from the StaffManager class to check on the staff's availability and provides an interface for the StaffManager class to reject jobs.

Studio Entity Class
The Studio class stores information about what types of instruments it contains for it to be allocated to a type of job. This Studio class has an aggregation relationship with the Job class. This class is also used by the StudioController control class for managing the studio.

StudioController Control Class
The StudioController class uses the Studio class and provides an interface to the ManagerController control class for managing the studios.

3.2.1 Entity Class Modeling

This section details entity class modeling to extract classes and their attributes by deducing them from use cases and their scenarios.

Step 1. Concise problem definition
Staff have to indicate availability and job preference in advance for managers to allocate jobs to them. During job allocation, managers must allocate a job at a studio and an instrument to the staff based on the staff's availability and qualification. The IT administrator manages the employee accounts of both the staff and managers.

Step 2. Identify the nouns
Staff have to indicate availability and job preference in advance for managers to allocate jobs to them. During job allocation, managers must allocate a job at a studio and an instrument to the staff based on the staff's availability and qualification. The IT administrator manages the employee accounts of both the staff and managers.

Candidate class:
Employee, Staff, Manager, Admin, Availability, Preference, Job, Studio, Instrument

3.2.2 Class Diagram

Inheritance

Employee

The system has different types of users that share the same attributes. These users inherit from the employee abstract class, which contains personal information as well as information used for authentication. As shown in Figure 8, this improves code modularity for code reusability as the different users have the same base properties.

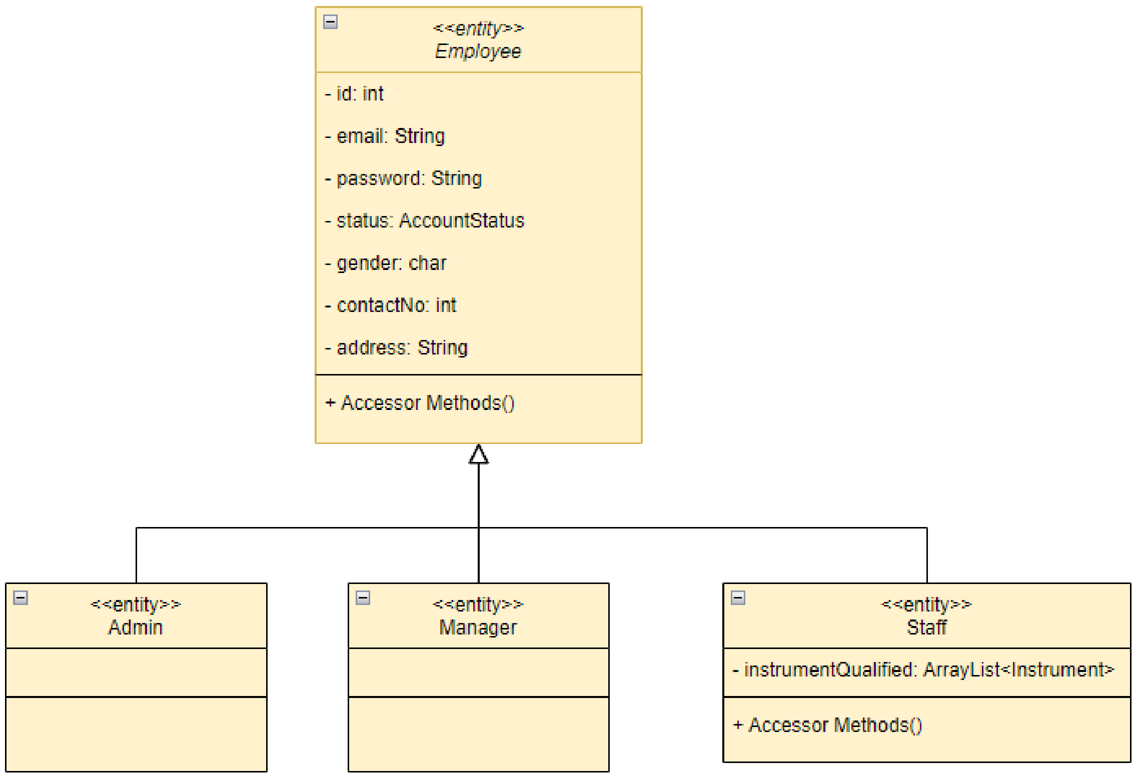
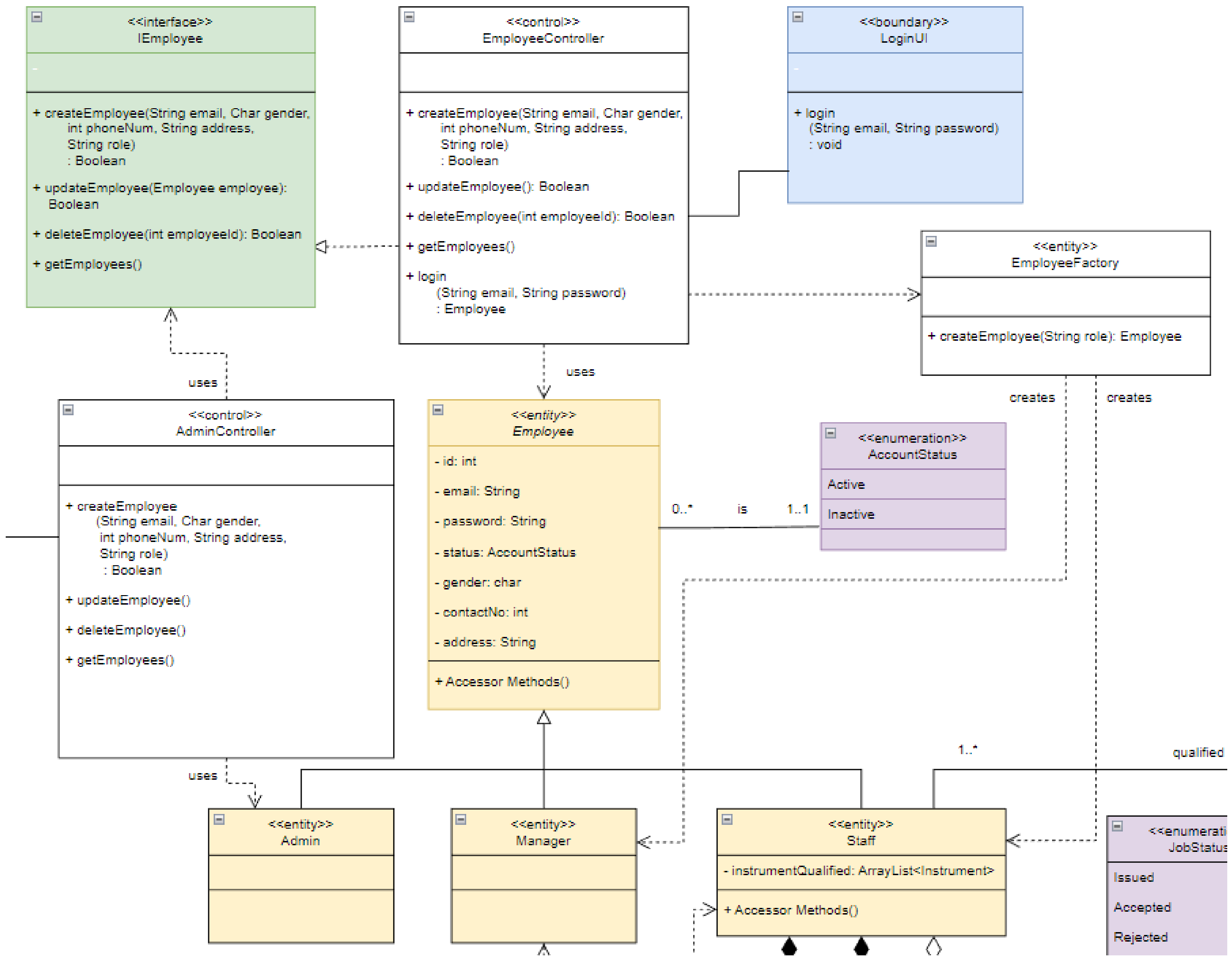


Figure 8: Employee Inheritance

Factory Design Pattern

Figure 9 illustrates how the EmployeeFactory returns a specific subclass based on the role that was passed in. This allows for future additions of other types of employees that may exist in this Music School System in the future.



<<interface>> IEmployee
+ createEmployee(String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee(Employee employee): Boolean
+ deleteEmployee(int employeeId): Boolean
+ getEmployees()
<<control>> EmployeeController
+ createEmployee(String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee(): Boolean
+ deleteEmployee(int employeeId): Boolean
+ getEmployees()
+ login (String email, String password) : Employee
<<boundary>> LoginUI
+ login (String email, String password) : void
<<entity>> EmployeeFactory
+ createEmployee(String role): Employee
uses
uses
<<control>> AdminController
+ createEmployee (String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee()
+ deleteEmployee()
+ getEmployees()
<<entity>> Employee
- id: int
- email: String
- password: String
- status: AccountStatus
- gender: char
- contactNo: int
- address: String
+ Accessor Methods()
creates
creates
<<enumeration>> AccountStatus
Active
Inactive
0..*
is
1..1
1..*
qualified
uses
<<entity>> Admin
<<entity>> Manager
<<entity>> Staff
- instrumentQualified: ArrayList<Instrument>
+ Accessor Methods()
<<enumerati JobStatus
Issued
Accepted
Rejected

Figure 9: Factory Design Pattern

3.2.3 Sequence Diagram

The following Figure 10-16 shows the sequence diagram which explains the design flow between objects for staff, managers, and IT administrator use cases.

Staff

View Overall Workload

Figure 10 shows the sequence diagram for the View Overall Workload use case where the staff chooses to view the overall workload. The overall workload data is extracted by the system upon staff login. The system returns the results from the database and displays them on the StaffDashboardUI.

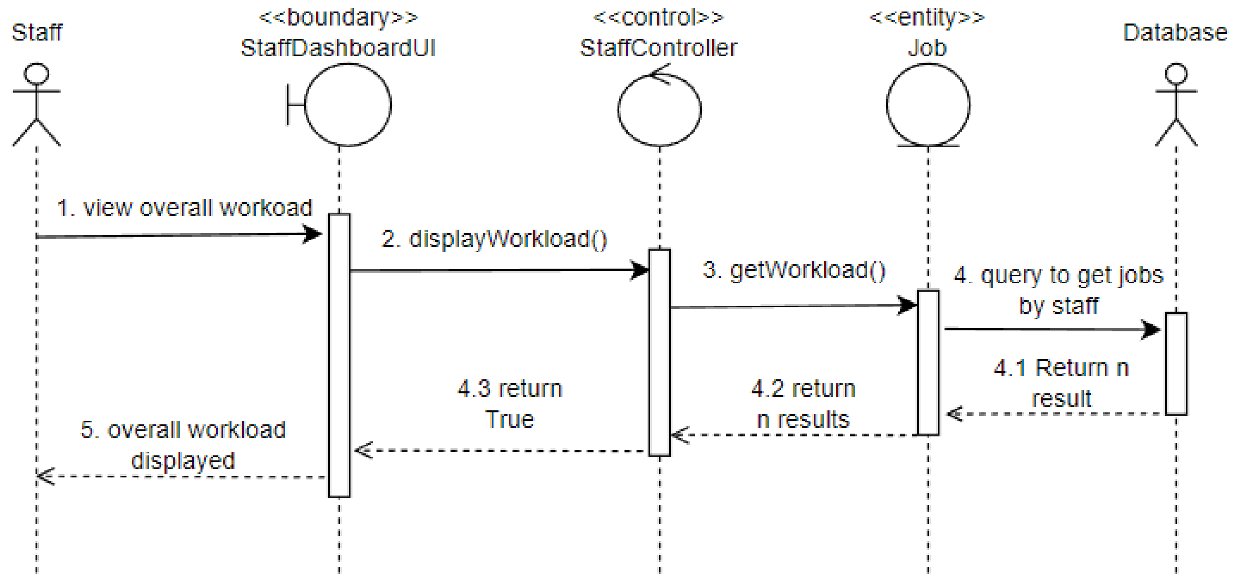


Figure 10: Staff - View Overall Workload Sequence Diagram

Reject Job Assigned

Figure 11 shows the sequence diagram for the Reject Job Assigned use case where the staff chooses to reject jobs assigned to them. The system would first display a warning message reminding staff to talk to their manager first before rejecting. After the staff have accepted the message, the system will update the database by changing the job status to ‘Rejected’ and display a confirmation message on the StaffDashboardUI. The alternate scenario includes the possibility of staff not being able to reject assigned jobs. Hence, the system will not update job status and return an error message with reasons on the StaffDashboardUI.

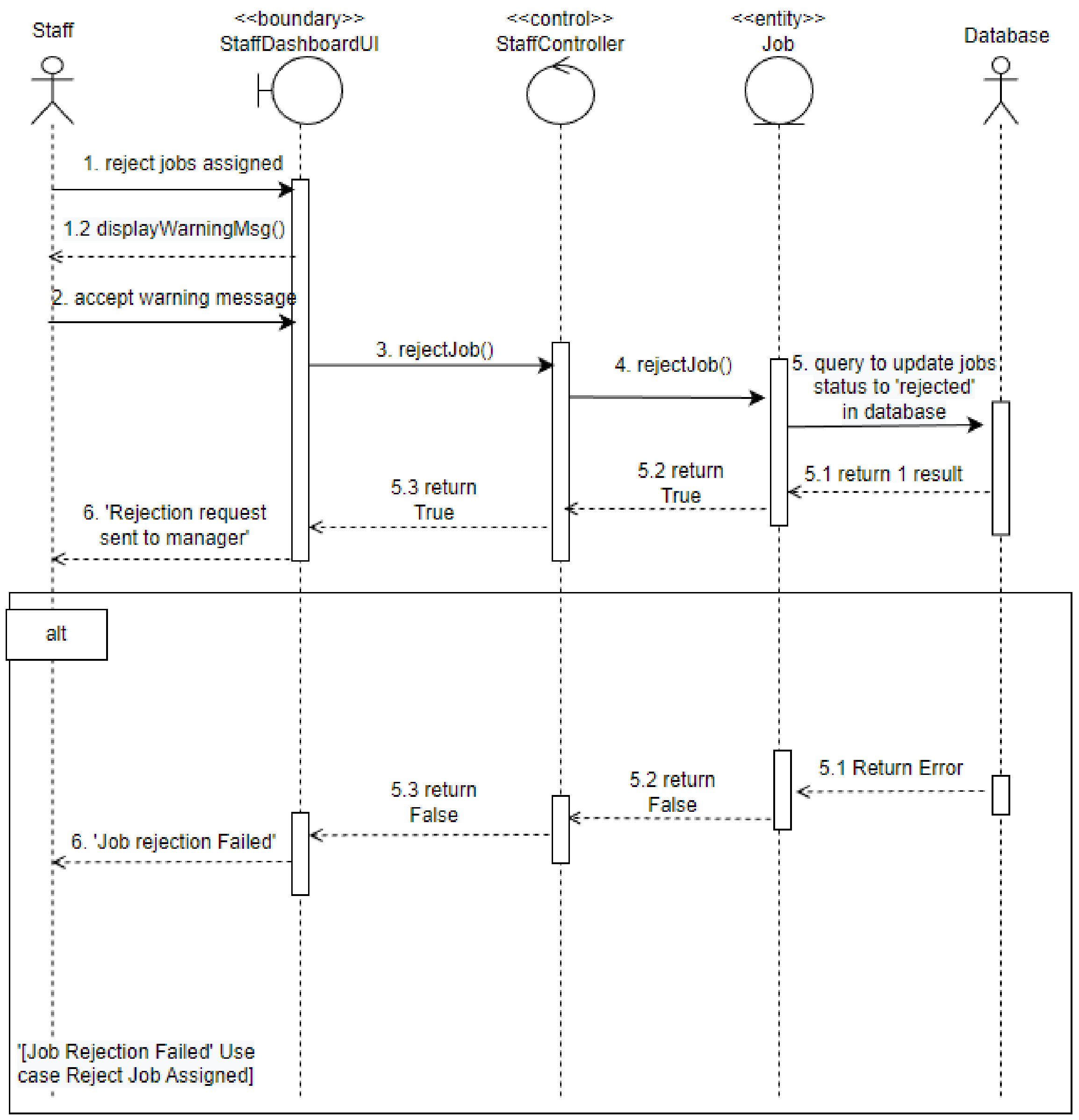


Figure 11: Staff - Reject Job Assigned Sequence Diagram

Add Availability

Figure 12 shows the sequence diagram for the Add Availability use case where the staff chooses to submit their availability. The system adds staff availability to the database and displays a confirmation message on the Staff addAvailability UI.

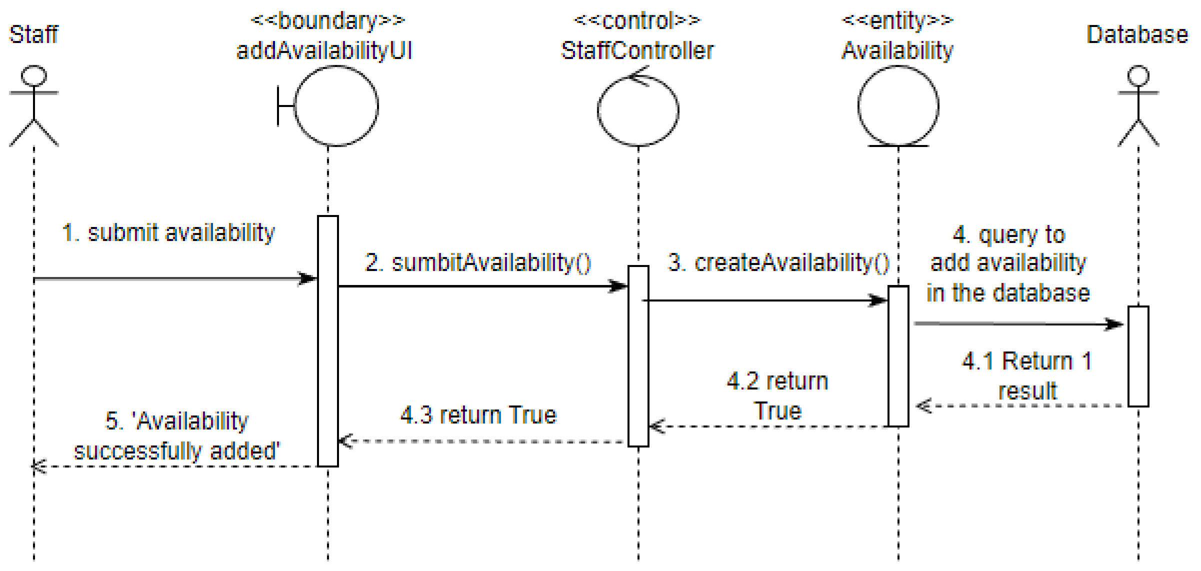


Figure 12: Staff - Add Availability Sequence Diagram

Indicate Job Preference

Figure 13 shows the sequence diagram for the Indicate Job Preference use case where the staff chooses to indicate their job preference. The system adds staff job preferences to the database and displays a confirmation message on the Staff addPreferences UI.

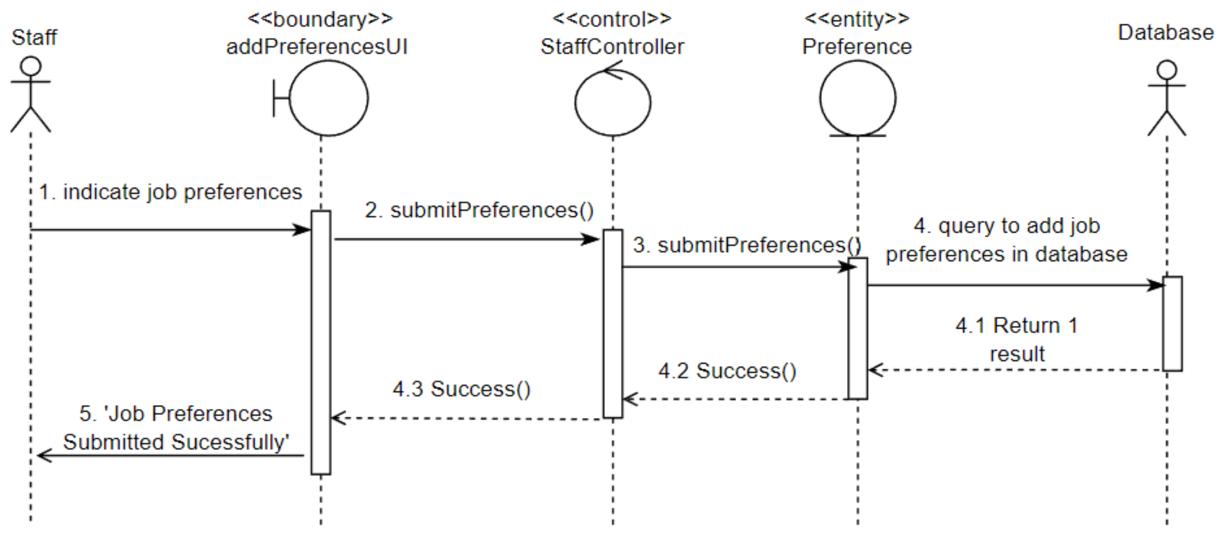


Figure 13: Staff - Indicate Job Preference Sequence Diagram

Manager

View Staff Workload

Figure 14 shows the sequence diagram for the View Staff Workload use case where the manager chooses to view the overall workload. The overall workload data is extracted by the system upon manager login. The system returns the results from the database and displays them on the MangerDashboardUI.

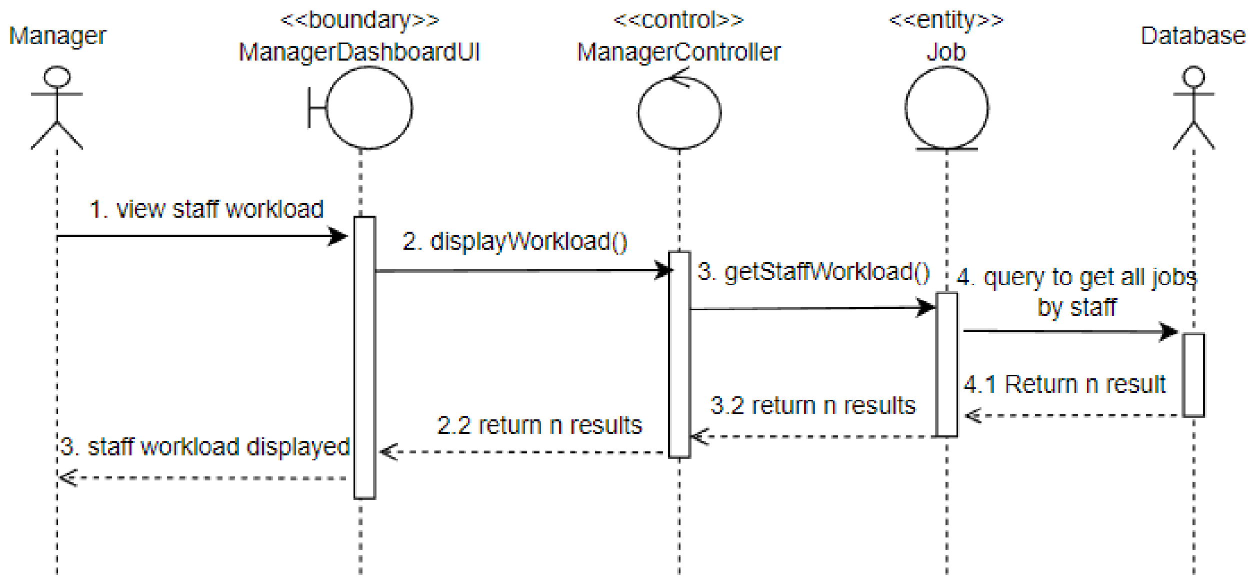


Figure 14: Manager - View Staff Workload Sequence Diagram

Allocate Job

Figure 15 shows the sequence diagram for the Allocate Job use case where the manager chooses to view staff availability before assigning the job on the AllocateJobUI. The system extracts the information from the database and displays the result on the AllocateJobUI. Afterward, the manager would need to select the instrument type, date, and time slot and obtain the available studio for the selected option. The system would extract the available studio from the database and display it on the AllocateJobUI. Finally, the system would validate and check the available studio as well as the staff availability before allowing the creation of jobs. The alternative scenarios include the possibility of not having an available studio and staff and also the number of hours for the particular staff has exceeded. In the mentioned scenarios, the manager will not be able to assign jobs.

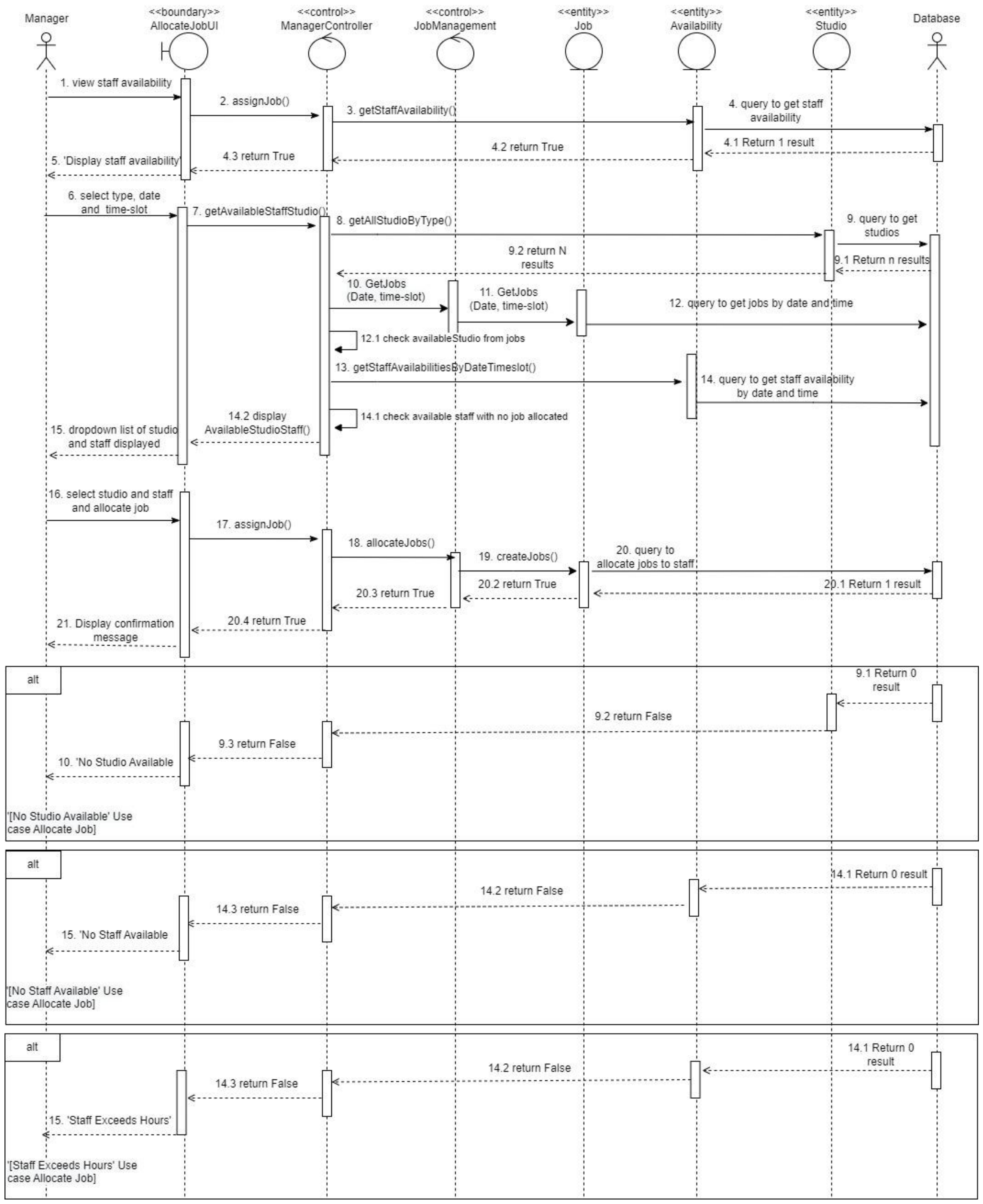

Manager
<<boundary>> AllocateJobUI
<<control>> ManagerController
<<control>> JobManagement
<<entity>> Job
<<entity>> Availability
<<entity>> Studio
Database
1. view staff availability
2. assignJob()
3. getStaffAvailability()
4. query to get staff availability
4.1 Return 1 result
4.2 return True
4.3 return True
5. 'Display staff availability'
6. select type, date and time-slot
7. getAvailableStaffStudio()
8. getAllStudioByType()
9. query to get studios
9.1 Return n results
9.2 return N results
10. GetJobs (Date, time-slot)
11. GetJobs (Date, time-slot)
12. query to get jobs by date and time
12.1 check availableStudio from jobs
13. getStaffAvailabilitiesByDateTimeslot()
14. query to get staff availability by date and time
14.1 check available staff with no job allocated
14.2 display AvailableStudioStaff()
15. dropdown list of studio and staff displayed
16. select studio and staff and allocate job
17. assignJob()
18. allocateJobs()
19. createJobs()
20. query to allocate jobs to staff
20.1 Return 1 result
20.2 return True
20.3 return True
20.4 return True
21. Display confirmation message
alt
9.1 Return 0 result
9.2 return False
9.3 return False
10. 'No Studio Available
'[No Studio Available' Use case Allocate Job]
alt
14.1 Return 0 result
14.2 return False
14.3 return False
15. 'No Staff Available
'[No Staff Available' Use case Allocate Job]
alt
14.1 Return 0 result
14.2 return False
14.3 return False
15. 'Staff Exceeds Hours'
'[Staff Exceeds Hours' Use case Allocate Job]

Figure 15: Manager - Allocate Job Sequence Diagram

IT Administrator

Create Account

Figure 16 shows the sequence diagram for the Create Account use case, where the IT Administrator chooses to create an account on the AccountManagementUI by submitting employee information. The system returns the confirmation results and they are displayed on the AccountManagementUI. The alternative scenario includes the possibility of registering the same email for different accounts. The same email is not allowed in the system, hence, the account will not be created.

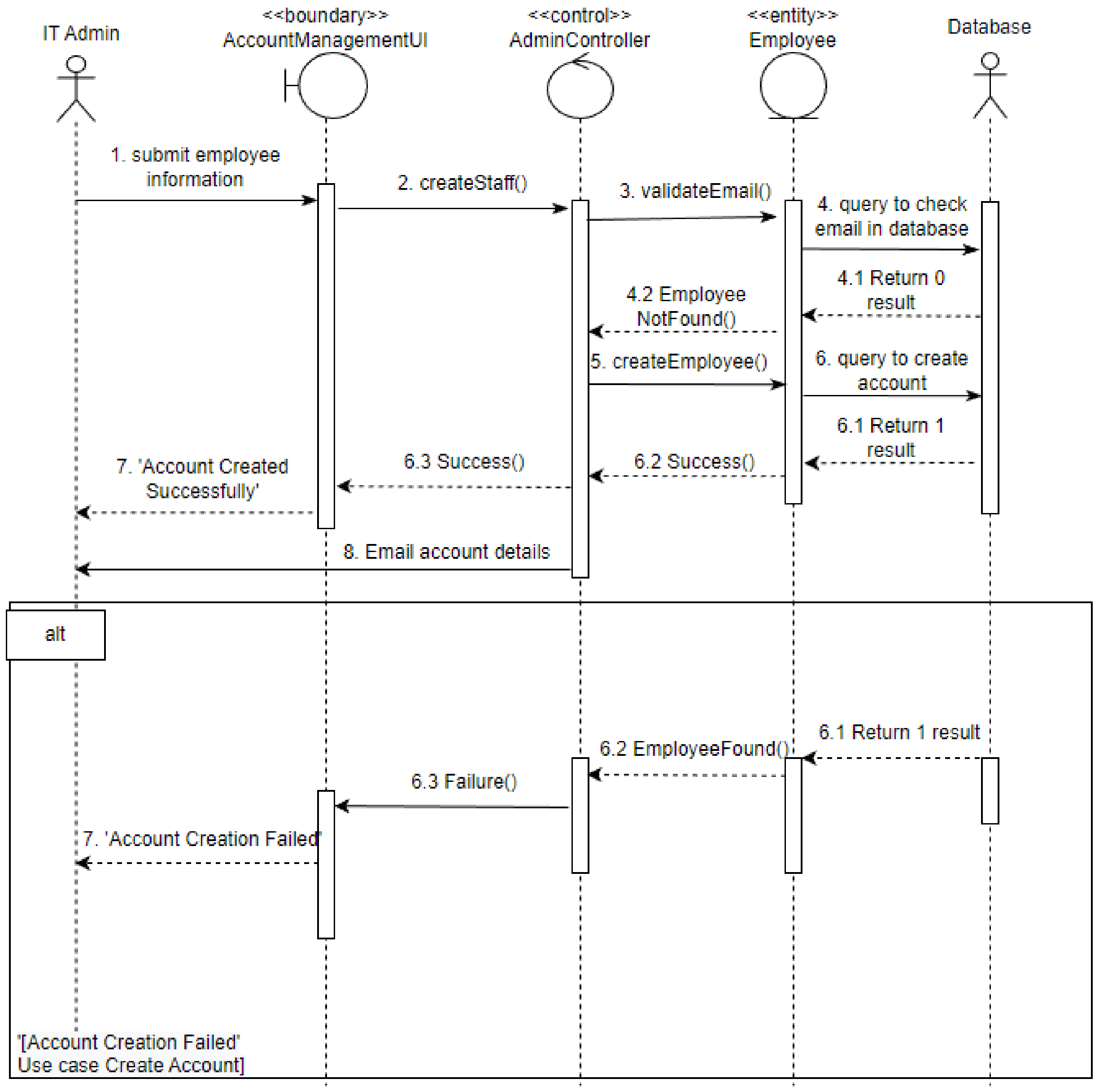


Figure 16: IT Administrator - Create Account Sequence Diagram

4 Project Plan

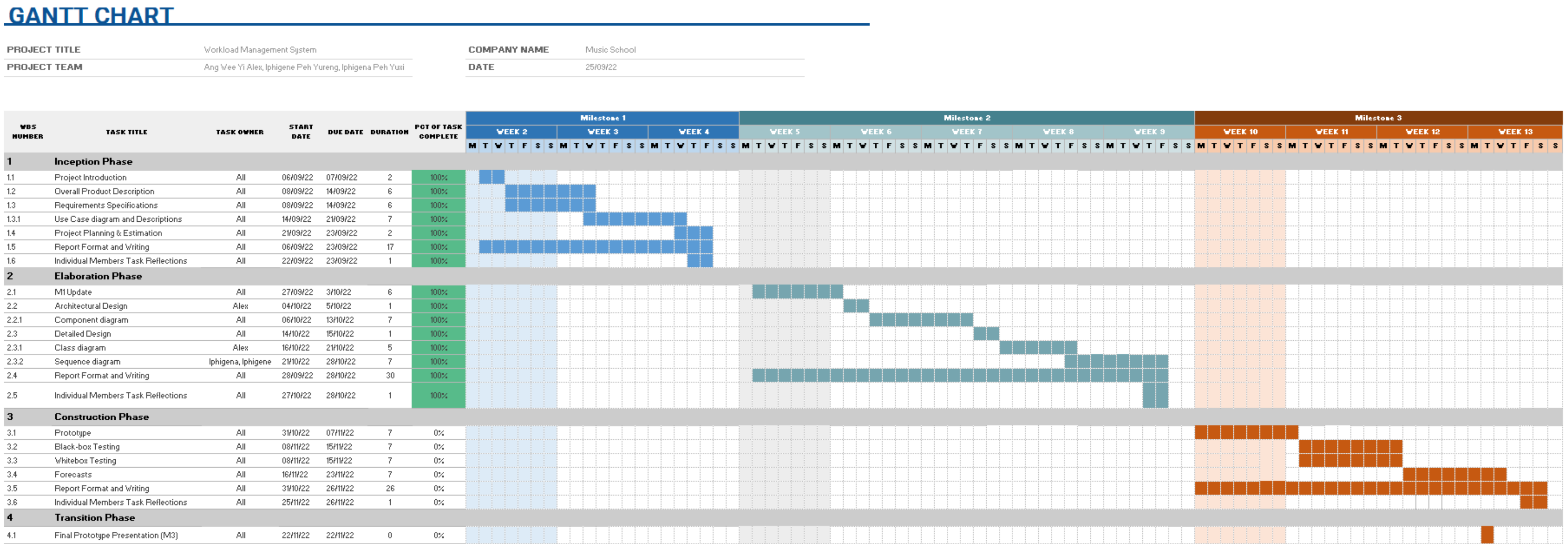
The project plan section would describe the Software Development Life Cycle (SDLC), resources required, and overall time required for the project via effort estimation and estimated time duration. The section would also include the current status of the project with a Gantt chart as well as the work allocation for this project via the work breakdown structure (WBS) diagram. In addition, an explanation of any overruns in the project will also be included.

4.1 SDLC

The software development life cycle chosen for this project is the Rational Unified Process (RUP) model, with the focus on delivering high quality software through object-oriented models, with their design and documentation expressed using the Unified Modelling Language (UML). This allows the business process to be tied to the development process and deliver value early to the client while focusing on the quality of design that is suitable for this project. In addition, it does not require close collaboration with the client, which the client prefers for this project, and our team has experience in object-oriented programming, which is beneficial for this SDLC model. In conclusion, with the strict timeline and budget that this project has, this model is the most suitable, as this project has three specific milestones that can be achieved using this model with the required specifications composed of many UML diagrams.

4.2 Updated Gantt Chart

Figure 17 shows the Gantt chart, which includes the current status of the project as well as the planned duration for Milestone 3.



GANTT CHART

PROJECT TITLE	Workload Management System	COMPANY NAME	Music School
PROJECT TEAM	Ang Wee Yi Alex, Iphigene Peh Yureng, Iphigena Peh Yuxi	DATE	25/09/22

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Inception Phase**					
1.1	Project Introduction	All	06/09/22	07/09/22	2	100%
1.2	Overall Product Description	All	08/09/22	14/09/22	6	100%
1.3	Requirements Specifications	All	08/09/22	14/09/22	6	100%
1.3.1	Use Case diagram and Descriptions	All	14/09/22	21/09/22	7	100%
1.4	Project Planning & Estimation	All	21/09/22	23/09/22	2	100%
1.5	Report Format and Writing	All	06/09/22	23/09/22	17	100%
1.6	Individual Members Task Reflections	All	22/09/22	23/09/22	1	100%
2	**Elaboration Phase**					
2.1	M1 Update	All	27/09/22	3/10/22	6	100%
2.2	Architectural Design	Alex	04/10/22	5/10/22	1	100%
2.2.1	Component diagram	All	06/10/22	13/10/22	7	100%
2.3	Detailed Design	All	14/10/22	15/10/22	1	100%
2.3.1	Class diagram	Alex	16/10/22	21/10/22	5	100%
2.3.2	Sequence diagram	Iphigena, Iphigene	21/10/22	28/10/22	7	100%
2.4	Report Format and Writing	All	28/09/22	28/10/22	30	100%
2.5	Individual Members Task Reflections	All	27/10/22	28/10/22	1	100%
3	**Construction Phase**					
3.1	Prototype	All	31/10/22	07/11/22	7	0%
3.2	Black-box Testing	All	08/11/22	15/11/22	7	0%
3.3	Whitebox Testing	All	08/11/22	15/11/22	7	0%
3.4	Forecasts	All	16/11/22	23/11/22	7	0%
3.5	Report Format and Writing	All	31/10/22	26/11/22	26	0%
3.6	Individual Members Task Reflections	All	25/11/22	26/11/22	1	0%
4	**Transition Phase**					
4.1	Final Prototype Presentation (M3)	All	22/11/22	22/11/22	0	0%

Figure 17: Gantt Chart

Estimation of Size

The process of size estimation is shown in the following steps 1-4.
Therefore, the calculation of Use Case Points (UCP) = UUCW * TCF * EF
= 101 * 1.0 * 0.77
= 77.77
= 78 (nearest whole number)

Step 1: Calculate unadjusted weight of use cases

Table 2 shows the calculation of unadjusted weight of use cases and is summarized as such:

- 12 simple and 3 average use cases
- Unadjusted weight of use cases = 13 (5) + 1 (10) + 1(15) = 90

Use Case Number	Use Case Name	Total Transactions/Use Case	Complexity of a Use Case
1	View Staff Workload	1	5
2	Allocate Job	6	10
3	Reallocate Job	8	15
4	View Staff Availability	2	5
5	View Assigned Jobs	1	5
6	View Overall Workload	1	5
7	Reject Job Assigned	3	5
8	Add Availability	2	5
9	Indicate Job Preference	2	5
10	Edit Availability	2	5
11	View Availability	2	5
12	View Accounts	2	5
13	Edit Accounts	2	5
14	Delete Accounts	2	5
15	Create Accounts	3	5
		Total unadjusted use case weight	**90**

Table 2: Unadjusted Weight of Use Case

Step 2: Identifying Actors

Table 3 shows the calculation of unadjusted weight of actors and is summarized as such:
- 1 average and 3 complex actors
- Unadjusted weight of actors = 1(2) + 3(3) = 11

Actor	weight	count	product
Simple	1	0	0
Medium	2	1	2
Complex	3	3	9
		Total	11

Table 3: Unadjusted Weight of Actors

Therefore, the Total Unadjusted Points (UUCW) = 90 + 11 = 101

Step 3: Determining the Technical Complexity Factors (TCF)

Table 4 shows the calculation of degree of influence for TCF and is summarized as such:
- Degree Of Influence (DI) = 40

Factor	Description	Weight	Assesment	Product
T1	Distributed system	2	0	0
T2	Response time/performance objectives	1	1	1
T3	End-user efficiency	1	2	2
T4	Internal processing complexity	1	3	3
T5	Code reusability	1	5	5
T6	Easy to install	0.5	4	2
T7	Easy to use	0.5	4	2
T8	Portability to other platforms	2	5	10
T9	System maintenance	1	4	4
T10	Concurrent/parallel processing	1	1	1
T11	Security features	1	5	5
T12	Access for third parties	1	0	0
T13	End user training	1	5	5
			Total	40

Table 4: Degree Of Influence (TCF)

Therefore, the TCF = 0.6 + 0.01 x 40 = 1.0

Step 4: Environment Factors (EF)

Table 5 shows the calculation of degree of influence for EF and is summarized as such:

- Degree Of Influence (DI) = 21

Environment	weight	Assessment	Product
Familiar with Development Process	1.5	4	6
Part time workers	-1	0	0
Analyst capability	0.5	3	1.5
Application experience	0.5	3	1.5
Object oriented experience	1	4	4
Motivation	1	3	3
Difficult programing language	-1	1	-1
Stable requirements	2	3	6
		Total	21

Table 5: Degree Of Influence (EF)

Therefore, the EF = 1.4 + (-0.03 x 21) = 0.77

Estimation of Effort

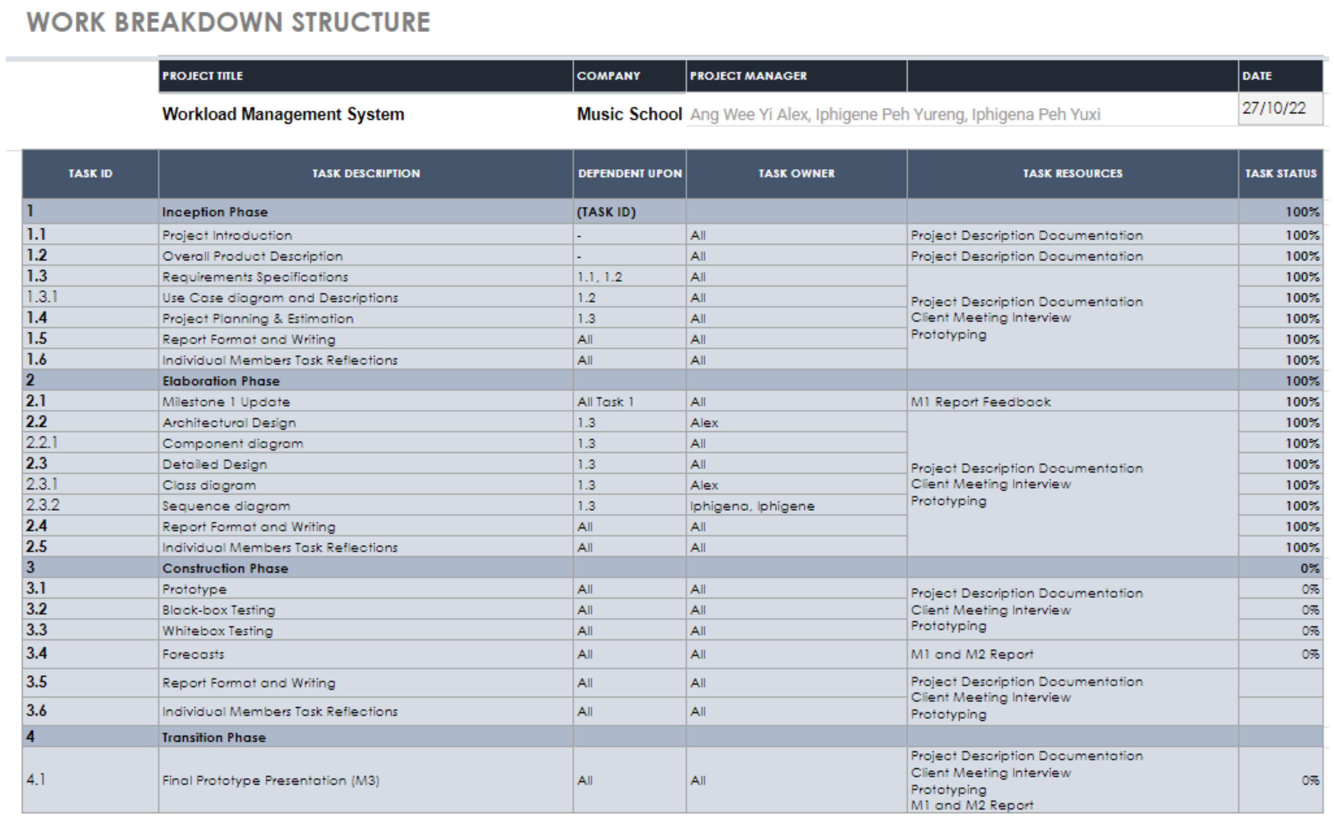
The effort was estimated using UCP where an estimation of 15-30 hours was given to each UCP. The estimated effort in developer-hours of a project of size 78 UCP = 15 * 78 to 30 * 78 hours = 1170 to 2340 hours

Assign Resources

The main resource identified for this project would be people power (i.e., time). Hence, the Gantt chart, WBS diagram, and task duration segment would be the plans to assign the main resource.

WBS diagram

Figure 18 illustrates the WBS for planned outcomes to reach the goal of the project.



WORK BREAKDOWN STRUCTURE

PROJECT TITLE	COMPANY	PROJECT MANAGER		DATE
Workload Management System	Music School	Ang Wee Yi Alex, Iphigene Peh Yureng, Iphigena Peh Yuxi		27/10/22

TASK ID	TASK DESCRIPTION	DEPENDENT UPON	TASK OWNER	TASK RESOURCES	TASK STATUS
1	**Inception Phase**	(TASK ID)			100%
1.1	Project Introduction	-	All	Project Description Documentation	100%
1.2	Overall Product Description	-	All	Project Description Documentation	100%
1.3	Requirements Specifications	1.1, 1.2	All	Project Description Documentation Client Meeting Interview Prototyping	100%
1.3.1	Use Case diagram and Descriptions	1.2	All		100%
1.4	Project Planning & Estimation	1.3	All		100%
1.5	Report Format and Writing	All	All		100%
1.6	Individual Members Task Reflections	All	All		100%
2	**Elaboration Phase**				100%
2.1	Milestone 1 Update	All Task 1	All	M1 Report Feedback	100%
2.2	Architectural Design	1.3	Alex	Project Description Documentation Client Meeting Interview Prototyping	100%
2.2.1	Component diagram	1.3	All		100%
2.3	Detailed Design	1.3	All		100%
2.3.1	Class diagram	1.3	Alex		100%
2.3.2	Sequence diagram	1.3	Iphigena, Iphigene		100%
2.4	Report Format and Writing	All	All		100%
2.5	Individual Members Task Reflections	All	All		100%
3	**Construction Phase**				0%
3.1	Prototype	All	All	Project Description Documentation Client Meeting Interview Prototyping	0%
3.2	Black-box Testing	All	All		0%
3.3	Whitebox Testing	All	All		0%
3.4	Forecasts	All	All	M1 and M2 Report	0%
3.5	Report Format and Writing	All	All	Project Description Documentation Client Meeting Interview Prototyping	
3.6	Individual Members Task Reflections	All	All		
4	**Transition Phase**				
4.1	Final Prototype Presentation (M3)	All	All	Project Description Documentation Client Meeting Interview Prototyping M1 and M2 Report	0%

Figure 18: WBS Diagram

Estimation of Task Duration

Based on the three point estimation model [5], the team has decided:

1. The Optimistic duration (OD) is 20 days.
2. The Pessimistic duration (PD) is 30 days.
3. The expected duration (ED) is 22 days.

Hence, the weighted average of the most likely duration (D) as follows:

$D = (1 \times \boldsymbol{OD}) + (4 \times \boldsymbol{ED}) + (1 \times \boldsymbol{PD})\ /6$

$D = (12 + (4 * 15) + 27) / 6$

$D = 23$ days

Project Overruns

The duration for the creation of individual diagrams such as architectural design, component diagram, sequence diagrams, and detailed design had overrun as new changes were made throughout the assignment period, specific requirements changes would also be reflected and directly affect the changes in the mentioned diagrams.

5 Individual Members Task Reflections

Iphigena Peh Yuxi
For milestone 2 (M2), I was in charge of ensuring everyone communicated with each other to clarify doubts and align opinions. Additionally, I have contributed to finishing the report, specific requirements, sequence and component diagrams, and improving certain sections such as the scope and use case descriptions based on M1 feedback. While working on M2, I gained a better understanding of architectural and design patterns and improved my ability to distinguish between different architectural and design patterns. I have also learned how to better use sequence diagrams which are useful in illustrating the flow of each use case.

If given another chance, I believe that the team could have arranged a consultation session earlier with the professor to clarify the doubts we have regarding M2. It would be more beneficial to the team as we would then be able to express our issues in a clear and concise manner compared to asking questions via email. Moreover, as a team, I feel that we have done a great job improving our communication as compared to M1. In addition, I suggest that we still work on asking better-phrased questions among ourselves to avoid wasting time due to misinterpretations.

Iphigene Peh Yureng
For milestone two, I was in charge of setting at least a day every week to have a meeting either online via discord or physically for a progress update on the tasks. In addition, I completed the updates from the milestone one report and project plan section, as well as various diagrams, such as the component and sequence diagrams. I learned the difference between each diagram presented in the report and how to use the mentioned diagrams to illustrate the sequence and flow of the use case descriptions. I learned the importance of drawing these diagrams as they are essential in depicting and giving a better overview of the software design.

I believe we can complete the diagrams earlier and more accurately at the beginning of the assignment, given a better understanding of the different diagrams and when to utilise the mentioned diagrams. As a team, we made significant improvements from milestone one and managed to clarify any questions we had when in doubt. For milestone three, I believe the team can further improve by arranging consultation sessions and planning buffer time for any overruns.

Ang Wee Yi, Alex
For milestone 2, I was in charge of identifying tasks that can be concurrently done to allocate to everyone. In addition, I contributed to the report and architectural design diagrams which includes the C4 diagram as well as various other component diagrams including the class diagram. By drawing these diagrams, I learnt the importance of communication over the details that the diagram is supposed to bring across. When designing the class diagram, I learnt to design classes to be modular and enhance

code-reusability through concepts like inheritance and future scalability by using factory design patterns.

When drawing these diagrams, I had many doubts on what notations to use and how the various different concepts link to each other such as the Entity Control Boundary pattern and interfaces between components. If I had another chance, I would arrange a meeting to clarify these doubts. As a team, we met up physically to discuss the project which allowed us to clarify doubts with each other before moving on which saved us more time than in milestone 1.

6 Appendix A – Updated Use Case Descriptions

Use Case ID:	UC-1
Use Case Name:	**View Staff Workload**
Description:	This View Staff Workload use case allows the Manager to view the staff workload on the landing page.
Primary Actor:	Manager
Preconditions:	Manager is logged into the system.
Postconditions:	Staff Workload was displayed successfully.
Main Success Scenarios:	1. System displays the workload in hours allocated to each type of instrument 2. System displays the top three staff with the lowest workload. 3. System displays staff with over 40 hours of jobs allocated.
Alternative Scenarios:	

Use Case ID:	UC-2
Use Case Name:	**Allocate Job**
Description:	The Allocate Job use case allows the Manager to allocate jobs to the staff.
Primary Actor:	Manager
Preconditions:	Manager is logged into the system.
Postconditions:	System allows managers to view staff assigned jobs after they have allocated the job.
Main Success Scenarios:	1. System displays the job allocation page. 2. Manager chooses to view staff availability. 3. System displays staff availability. 4. Manager selects the type of lesson. 5. Manager selects the date and time slots. 6. System fetch available studio and staff. 7. System display dropdown list of studio and staff. 8. Manager selects the studio. 9. Manager selects the staff. 10. Manager submits the information. 11. System adds the job into the database, and displays a confirmation message.
Alternative Scenarios:	7a. No studios available for allocation. 7a1. System displays an error message that no studio is available for the selected time slot 7b. No staff available for allocation 7b1. System displays an error message that no staff is available for the selected time slot 8a. Staff exceeds the four hour consecutive work without 1 hour of rest 8a1. System displays an error message that the staff has exceeded the workload limit without rest.

Use Case ID:	UC-3
Use Case Name:	**Reallocate Job**
Description:	The Reallocate Job use case allows the Manager to only reallocate the jobs that were rejected by the staff. Hence, the <<extend>> relationship with the Allocate Job use case.
Primary Actor:	Manager
Preconditions:	Manager is logged into the system. Staff has rejected the job that was previously assigned by the manager.
Postconditions:	The rejected job is reallocated successfully.
Main Success Scenarios:	1. System reflects the rejected jobs in the database. 2. Manager chooses to reallocate the rejected job(s). 3. Manager selects the job to allocate. 4. System displays staff availability 5. Manager allocates the job based on staff availability. 6. System updates the job in the database, and displays a confirmation message.
Alternative Scenarios:	4a. No staff is available for the manager to assign the job. 4a1. System displays an error message saying there is no staff available 6a. Staff exceeds the four hour consecutive work without 1 hour of rest 6a1. System displays an error message that the staff has exceeded the workload limit without rest.

Use Case ID:	UC-4
Use Case Name:	**View Staff Availability**
Description:	The View Staff Availability use case allows the Manager to view the staff availability for job allocation.
Primary Actor:	Manager
Preconditions:	Manager is logged into the system.
Postconditions:	Staff availability was displayed successfully. System allows managers to view staff preference after they have viewed the staff availability.
Main Success Scenarios:	1. System extracts the list of staff availability, assigned jobs and job preference from the database. 2. System displays the list of staff availability, assigned jobs and job preference.
Alternative Scenarios:	1a. No staff availability is submitted into the database. 1a1. System displays an error message saying no staff available and provides the reason.

Use Case ID:	UC-5
Use Case Name:	**View Assigned Jobs**

Description:	The View Assigned Jobs use case allows the Staff to view the jobs assigned by the manager.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system. Manager has already assigned the job(s) to the staff.
Postconditions:	Assigned jobs are displayed successfully.
Main Success Scenarios:	1. System extracts the list of jobs assigned to the staff. 2. System displays the assigned jobs
Alternative Scenarios:	

Use Case ID:	UC-6
Use Case Name:	**View Overall Workload**
Description:	The View Overall Workload use case allows the staff to view their workload for the current month.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system. System has reflected jobs assigned by managers.
Postconditions:	Overall workload for the staff is displayed successfully.
Main Success Scenarios:	1. System extracts the list of jobs assigned to the staff. 2. System displays staff workload for the current month.
Alternative Scenarios:	

Use Case ID:	UC-7
Use Case Name:	**Reject Job Assigned**
Description:	The Reject Job Assigned use case allows the Staff to reject the job assigned by the manager.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system. Job was previously assigned by managers and recorded into the database.
Postconditions:	Job assigned to staff is rejected successfully.
Main Success Scenarios:	1. System extracts the jobs assigned to the staff from the database. 2. System displays jobs assigned to the staff. 3. Staff selects the job to reject. 4. System display warning message to staff 5. System record rejected jobs into the database.
Alternative Scenarios:	3a. Unable to reject job assigned by manager 3a1. System displays an error message saying unable to reject the job and provides the reason.

Use Case ID:	UC-8

Use Case Name:	**Add Availability**
Description:	The Add Availability use case allows the staff to add their availability up to 5 weeks ahead of time.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system.
Postconditions:	Staff availability was indicated in the system successfully. System allows staff to indicate their job preference.
Main Success Scenarios:	1. System displays the available time slots in 30 mins intervals. 2. Staff selects the time slots. 3. Staff submit information. 4. System displays a confirmation message.
Alternative Scenarios:	

Use Case ID:	UC-9
Use Case Name:	**Indicate Job Preference**
Description:	The Indicate Job Preference use case allows the staff to indicate the instruments they want to teach for the week if they have more than one instrument qualification. If the staff only has one qualification, they will not be allowed to indicate their job preference. Staff indicates their preference while adding their availability. Hence, the <<extend>> relationship with the Add Availability use case.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system. Staff have more than one instrument qualification. Staff is adding their availability.
Postconditions:	Job preference is indicated in the system successfully.
Main Success Scenarios:	1. System displays the instruments according to the staff qualification. 2. Staff selects the instrument. 3. Staff submits information. 4. System displays a confirmation message.
Alternative Scenarios:	

Use Case ID:	UC-10
Use Case Name:	**Edit Availability**
Description:	The Edit Availability use case allows the staff to edit their availability up to 5 weeks ahead of time.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system. Staff has viewed their availability.
Postconditions:	Staff availability was updated in the system successfully. System allows staff to indicate their job preference.

Main Success Scenarios:	1. Staff selects the modify option next to the specific date. 2. System displays a popup with time slots selected previously. 3. Staff modifies the information. 4. Staff submit information. 5. System modifies the availability in the database, and displays a confirmation message.
Alternative Scenarios:	

Use Case ID:	UC-11
Use Case Name:	**View Availability**
Description:	The View Availability use case allows the staff to view their availability up to 5 weeks ahead of time.
Primary Actor:	Staff
Preconditions:	Staff is logged into the system.
Postconditions:	Staff availability was displayed successfully.
Main Success Scenarios:	1. System extracts their availability from the database. 2. System displays their availability.
Alternative Scenarios:	2a. Staff did not submit availability. 2a1. System displays a message saying no availability was added.

Use Case ID:	UC-12
Use Case Name:	**View Accounts**
Description:	The View Accounts use case allows the IT administrators to manage staff and manager accounts by viewing the accounts.
Primary Actor:	IT administrator
Preconditions:	IT administrator is logged into the system.
Postconditions:	Accounts are displayed successfully.
Main Success Scenarios:	1. System extracts a list of accounts from the database. 2. System displays the list of accounts available.
Alternative Scenarios:	1a. No accounts are created for the IT administrator to view. 1a1. System displays an error message saying no accounts created.

Use Case ID:	UC-13
Use Case Name:	**Edit Accounts**
Description:	The Edit Accounts use case allows the IT administrators to manage staff and manager accounts by editing the account information.
Primary Actor:	IT administrator
Preconditions:	IT administrator is logged into the system. IT administrator has viewed the list of accounts created.

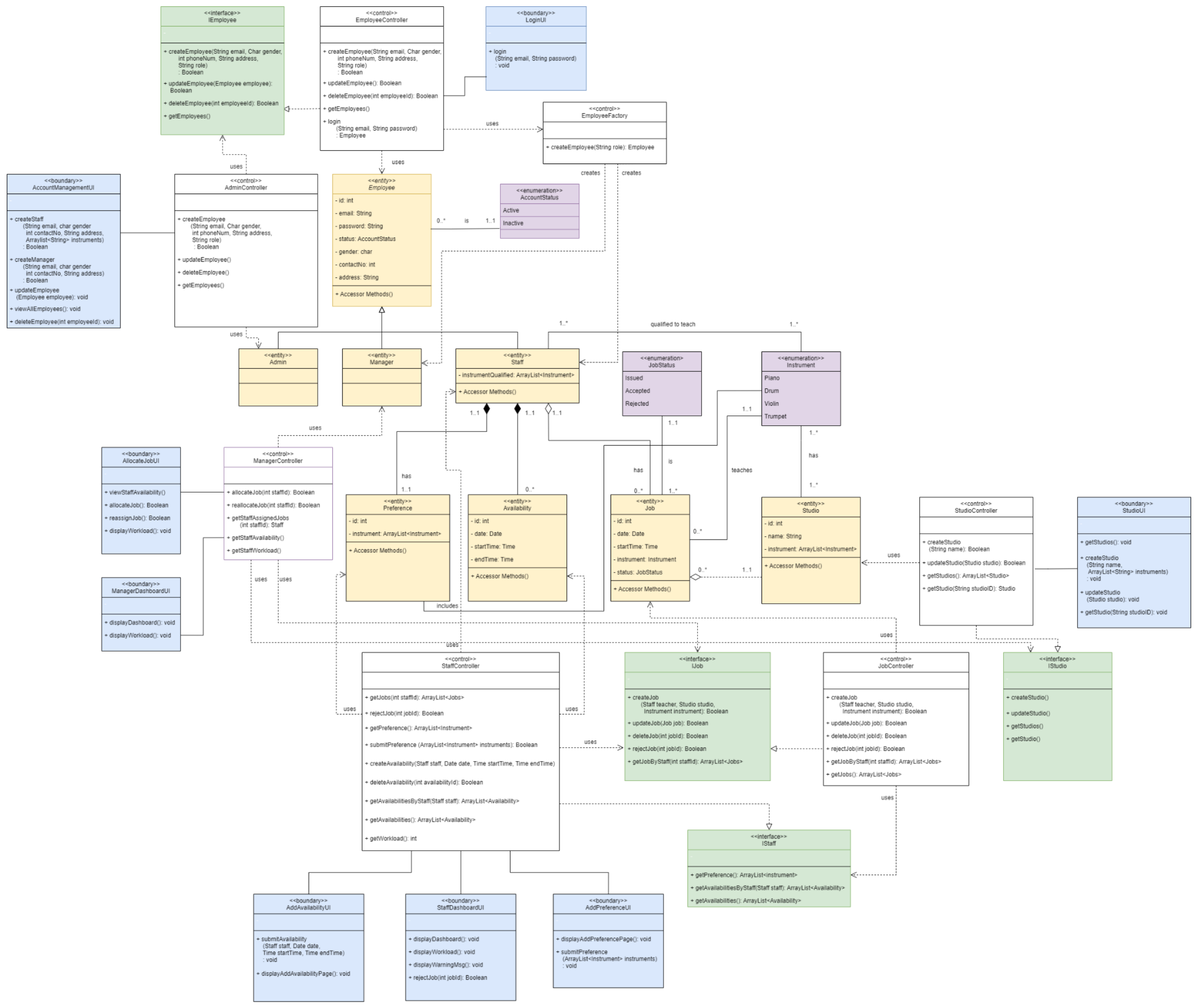
Postconditions:	Account was modified successfully.
Main Success Scenarios:	1. IT administrator selects the edit option next to the account. 2. System displays a popup with details about the selected account. 3. IT administrator edits the information. 4. IT administrator submits the information. 5. System modifies the account information in the database, and displays a confirmation message.
Alternative Scenarios:	

Use Case ID:	UC-14
Use Case Name:	**Delete Accounts**
Description:	The Delete Accounts use case allows the IT administrator to manage the staff and managers account by deleting the accounts.
Primary Actor:	IT administrator
Preconditions:	IT administrator is logged into the system. IT administrator has viewed the list of accounts created.
Postconditions:	Account was deleted successfully.
Main Success Scenarios:	1. IT administrator selects the delete option next to the account. 2. System prompts for delete account confirmation. 3. IT administrator confirms by clicking delete. 4. System deletes the account in the database, and displays a confirmation message.
Alternative Scenarios:	

Use Case ID:	UC-15
Use Case Name:	**Create Accounts**
Description:	The Create Accounts use case allows the IT administrator to manage the staff and managers account by creating the accounts.
Primary Actor:	IT administrator
Preconditions:	IT administrator is logged into the system.
Postconditions:	Account was created successfully.
Main Success Scenarios:	1. IT administrator selects either “Manager” or “Staff”. 2. IT administrator enters the staff/manager ID. 3. IT administrator enters the staff/manager name. 4. IT administrator enters the staff/manager email. 5. IT administrator enters the staff/manager contact information. 6. If creating a staff account, IT administrator enters the qualification. 7. IT administrator submit information.

	8. System adds the account into the database, and displays a confirmation message. 9. System sends an email to the account's email with the login details of the account
Alternative Scenarios:	4a. Email already exists. 4a1. System displays an error message saying the email already exists.

7 Appendix B - Overall Class Diagram


<<interface>> IEmployee
+ createEmployee(String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee(Employee employee): Boolean
+ deleteEmployee(int employeeId): Boolean
+ getEmployees()
<<control>> EmployeeController
+ createEmployee(String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee(): Boolean
+ deleteEmployee(int employeeId): Boolean
+ getEmployees()
+ login (String email, String password) : Employee
<<boundary>> LoginUI
+ login (String email, String password) : void
<<control>> EmployeeFactory
+ createEmployee(String role): Employee
uses
creates
creates
<<boundary>> AccountManagementUI
+ createStaff (String email, char gender int contactNo, String address, Arraylist<String> instruments) : Boolean
+ createManager (String email, char gender int contactNo, String address) : Boolean
+ updateEmployee (Employee employee): void
+ viewAllEmployees(): void
+ deleteEmployee(int employeeId): void
<<control>> AdminController
+ createEmployee (String email, Char gender, int phoneNum, String address, String role) : Boolean
+ updateEmployee()
+ deleteEmployee()
+ getEmployees()
<<entity>> Employee
- id: int
- email: String
- password: String
- status: AccountStatus
- gender: char
- contactNo: int
- address: String
+ Accessor Methods()
<<enumeration>> AccountStatus
Active
Inactive
0..* is 1..1
<<entity>> Admin
<<entity>> Manager
<<entity>> Staff
- instrumentQualified: ArrayList<Instrument>
+ Accessor Methods()
1..* qualified to teach 1..*
<<enumeration> JobStatus
Issued
Accepted
Rejected
<<enumeration>> Instrument
Piano
Drum
Violin
Trumpet
<<boundary>> AllocateJobUI
+ viewStaffAvailability()
+ allocateJob(): Boolean
+ reassignJob(): Boolean
+ displayWorkload(): void
<<boundary>> ManagerDashboardUI
+ displayDashboard(): void
+ displayWorkload(): void
<<control>> ManagerController
+ allocateJob(int staffId): Boolean
+ reallocateJob(int staffId): Boolean
+ getStaffAssignedJobs (int staffId): Staff
+ getStaffAvailability()
+ getStaffWorkload()
1..1 has 1..1
1..1 0..*
1..1 has 0..*
1..1 is 1..*
1..1 teaches 0..*
1..* has 1..*
0..* 1..1
includes
<<entity>> Preference
- id: int
- instrument: ArrayList<Instrument>
+ Accessor Methods()
<<entity>> Availability
- id: int
- date: Date
- startTime: Time
- endTime: Time
+ Accessor Methods()
<<entity>> Job
- id: int
- date: Date
- startTime: Time
- instrument: Instrument
- status: JobStatus
+ Accessor Methods()
<<entity>> Studio
- id: int
- name: String
- instrument: ArrayList<Instrument>
+ Accessor Methods()
<<control>> StudioController
+ createStudio (String name): Boolean
+ updateStudio(Studio studio): Boolean
+ getStudios(): ArrayList<Studio>
+ getStudio(String studioID): Studio
<<boundary>> StudioUI
+ getStudios(): void
+ createStudio (String name, ArrayList<String> instruments) : void
+ updateStudio (Studio studio): void
+ getStudio(String studioID): void
<<control>> StaffController
+ getJobs(int staffId): ArrayList<Jobs>
+ rejectJob(int jobId): Boolean
+ getPreference(): ArrayList<Instrument>
+ submitPreference (ArrayList<Instrument> instruments): Boolean
+ createAvailability(Staff staff, Date date, Time startTime, Time endTime)
+ deleteAvailability(int availabilityId): Boolean
+ getAvailabilitiesByStaff(Staff staff): ArrayList<Availability>
+ getAvailabilities(): ArrayList<Availability>
+ getWorkload(): int
<<interface>> IJob
+ createJob (Staff teacher, Studio studio, Instrument instrument): Boolean
+ updateJob(Job job): Boolean
+ deleteJob(int jobId): Boolean
+ rejectJob(int jobId): Boolean
+ getJobByStaff(int staffId): ArrayList<Jobs>
<<control>> JobController
+ createJob (Staff teacher, Studio studio, Instrument instrument): Boolean
+ updateJob(Job job): Boolean
+ deleteJob(int jobId): Boolean
+ rejectJob(int jobId): Boolean
+ getJobByStaff(int staffId): ArrayList<Jobs>
+ getJobs(): ArrayList<Jobs>
<<interface>> IStudio
+ createStudio()
+ updateStudio()
+ getStudios()
+ getStudio()
<<interface>> IStaff
+ getPreference(): ArrayList<instrument>
+ getAvailabilitiesByStaff(Staff staff): ArrayList<Availability>
+ getAvailabilities(): ArrayList<Availability>
<<boundary>> AddAvailabilityUI
+ submitAvailability (Staff staff, Date date, Time startTime, Time endTime) : void
+ displayAddAvailabilityPage(): void
<<boundary>> StaffDashboardUI
+ displayDashboard(): void
+ displayWorkload(): void
+ displayWarningMsg(): void
+ rejectJob(int jobId): Boolean
<<boundary>> AddPreferenceUI
+ displayAddPreferencePage(): void
+ submitPreference (ArrayList<Instrument> instruments) : void
uses

8 Appendix C – Data Dictionary

MongoDB

MongoDB is a source-available cross-platform document-oriented database program which uses JSON-like documents with optional schemas.

ExpressJS

Express.js is a back end web application framework, designed for building web applications and RESTful APIs with Node.js. It is released as free and open-source software under the MIT License.

NodeJS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine and executes JavaScript code outside a web browser.

AngularJS

AngularJS is a discontinued free and open-source JavaScript-based web framework for developing single-page applications.

Application Programming Interface

Application Programming Interface (API) is a set of definitions and protocols for building and integrating application software.

AES256

256-bit AES encryption is a technique that uses a key length of 256 bits for this process. The AES-256 key has the mathematical equivalent of 2256 potential combinations because key combinations increase exponentially with key size. Using 256-bit AES encryption assures the security of your data at rest.

2FA

Two-factor authentication (2FA) is a security mechanism for identity and access management that requires two forms of identification to access services and data.

MFA

Multi-factor authentication is a type of electronic authentication in which a user is only permitted access to a website or application after successfully providing two or more pieces of evidence to an authentication mechanism.

SSL

Secure Sockets Layer (SSL), is an encryption-based Internet security protocol. It was first developed by Netscape in 1995 to ensure privacy, authentication, and data integrity in Internet communications. SSL is the predecessor to the modern TLS encryption used today.

Denial of Service

Denial of service (DoS) is a type of cyber attack that aims to disable, shut down or disrupt a network, website or service.

UR

User Interface Requirement (UR) includes content presentation, application navigation, and user assistance of the product.

PF

Product Functionality (PR) is the functions that the product must perform or must let the user perform.

FR

Functional Requirements (FR) captures the intended behaviour of the system. This behavior may be expressed as services, tasks or functions the system is required to perform.

NFR

Non-functional Requirements (FR) includes generic properties of the system that focuses on system usability.