

ICS673F14 Software Engineering
Group Project 2 - Keyboard Application
Software Design Document

Your project Logo
here if any

[Revision History:](#)

[Introduction](#)

[Software Architecture](#)

[Architecture](#)

[Backend Overview](#)

[Backend Software Architecture](#)

[Backend Source Control Structure](#)

[Frontend Overview](#)

[Frontend Architecture](#)

[Frontend Directory Structure](#)

[Design Patterns](#)

[Key Algorithms](#)

[Classes and Methods](#)

[RESTful APIpassword](#)

[/user](#)

[GET /user](#)

[POST /user](#)

[/quiz](#)

[GET /quiz/user/<id>](#)

[GET /quiz/<id>](#)

[/tutorial_page?tutorialId=<id>](#)

[GET /tutorial_page?tutorialId=<id>](#)

[/tutorial_page/<id>pass](#)

[GET /tutorial_page/<id>](#)

[/tutorial_page_response/<id>](#)

[GET /tutorial_page_resopnse/<id>](#)

[/tutorial](#)

[GET /tutorial](#)

[Glossary](#)

<u>Team Member</u>	<u>Role(s)</u>	<u>Signature</u>	<u>Date</u>
Nigel Stuart Lead	<ul style="list-style-type: none">• Section 1 - Introduction<ul style="list-style-type: none">◦ Purpose• Section 2 - Backend Architecture<ul style="list-style-type: none">◦ Backend overview◦ Backend Architecture Diagram◦ Source Code Structure• Section 3 - Design Patterns (Backend)	<i>Nigel Stuart</i>	10/15/2014

	<ul style="list-style-type: none"> • Quiz and Demo REST API • Section 4 - Key Algorithms 		
Ana Beglova Configuration Leader	<ul style="list-style-type: none"> • Section 2 - Software Architecture (UI) • Section 3 - Design Patterns (UI) 		
Jonathan Kelley Backup Team Lead Implementation Lead	<ul style="list-style-type: none"> • Section 5 • Tutorial REST API 		
Christopher Wright Design	<ul style="list-style-type: none"> • Scope 		

Revision History:

<u>Version</u>	<u>Author</u>	<u>Date</u>	<u>Change</u>
1.0	Nigel Stuart	10/15/2014	Added RESTful API design items
1.1	Jonathan Kelley	10/15/2014	Added Current UML and Database Model diagrams
1.2	Nigel Stuart	12/6/2014	Added Backend Architecture sections. Added introduction section Added algorithms section.

1. Introduction

1.1 Purpose

This document contains the software architectural design for the virtual piano application called Noteable. Noteable is an online musical keyboard application allows users to learn how to read music and play notes online in real-time. The consists of tutorials in a structured manner, which slowly progress in difficult as the user completes levels. Each tutorial will be a building block, which will prepare users for the grande finale level, which includes playing the song “Happy Birthday” as well as “Row Row Row Your Boat”. The application will consist of tutorials, quizzes and demos which are intended to guide the student through the fundamentals of reading and applying music notation to a piano.

1.2 Scope

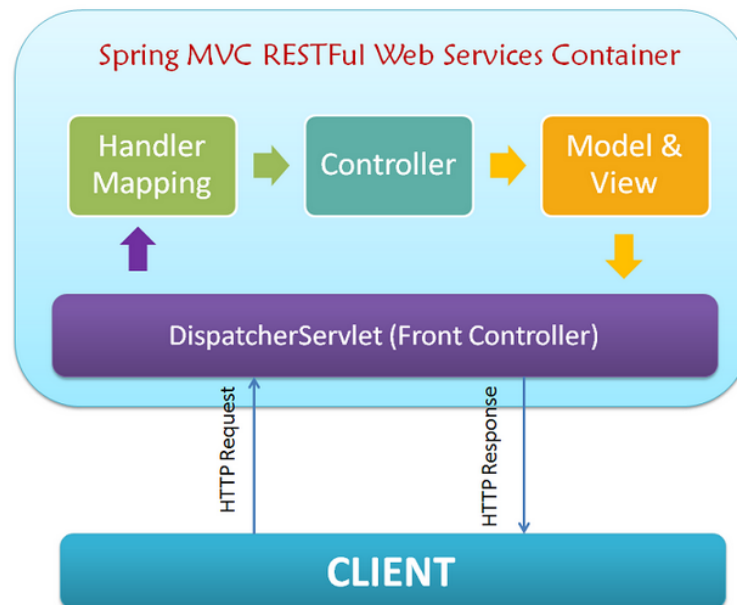
As stated above, the SDD outlines the aspects of engineering applied to building a piano-web application, named *NoteAble*. It is intended for students of any age who wish to learn the basics of note reading and apply those fundamentals to a piano keyboard. This tool can be used by students and instructors as part of lesson activities. The goal is to have an application which presents a note reading method in the form of tutorials and games. The benefits of this project include reinforced note and sight reading skills while keeping the students engaged. The tutorials apply mostly to a piano interface, but can be used as a note reading tool for many additional instruments.

2. Software Architecture

This section describes the decomposition of the *Noteable* software system, which include each component and the relationship between components.

a. Architecture

This section details the architecture for the *Noteable* keyboard application. The application consists of two separate architectures; a frontend and a backend. Communication is carried out between the two systems via RESTful API requests. For details about each section please see the below subsections. Below is an example of the architecture which Spring provides out of box for developers via annotations.

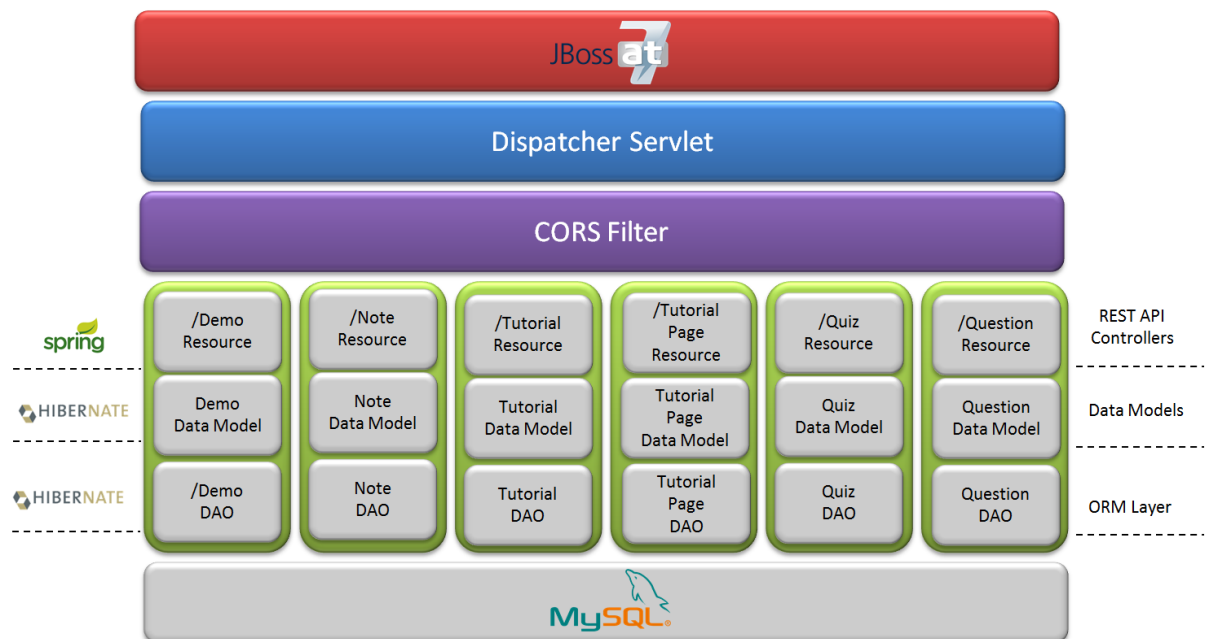


i. Backend Overview

The sole purpose of the backend is to provide a service which will complete data requests via HTTP to the frontend as needed. The backend will persist information within a MySQL database and will utilize Spring and Hibernate Java components to enable the service.

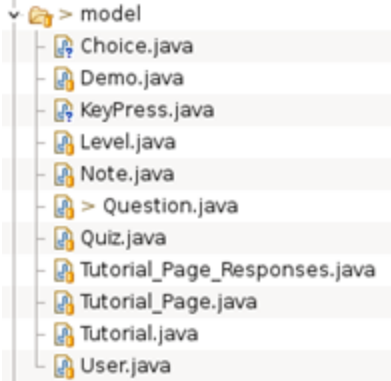
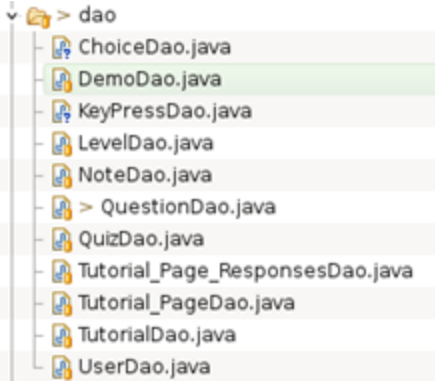
ii. Backend Software Architecture

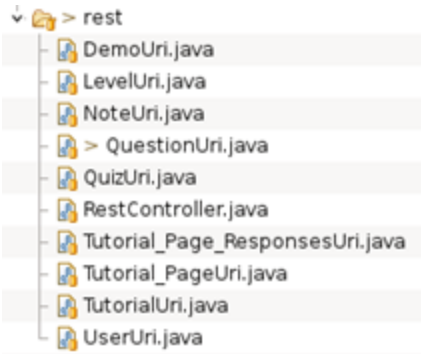
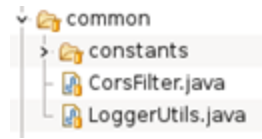
The backend service will provide services to send data related to Demos, Tutorials and Quizzes to the front end. Below is a diagram of the Architecture of the MusicService backend. Note each resource (in green) is a standalone resource which has an entry point via a restful call. Each entry point will enter the system and go through the Spring Data Model and DAO layer to interact with the database. It will then retrace its steps back up to the Resource Controller to return the requested data to the end user.



iii. Backend Source Control Structure

To keep the backend source code organized, it shall be broken up into five main categories, [model, common, dao, uri, test and SQL] each of which are described in the below table.

Models	<p>Used to maintain a standardized data structure per noun used within the project.</p> 
Data Access Objects (DAO)	<p>An abstraction layer located directly above the database, which is used to persist and collect information from the database. <i>Noteable</i> will be using the Hibernate API to communicate with the database. By having this abstraction layer it allows for the underlying database to be changed (i.e.: From MySQL to Oracle) without requiring code modifications to occur. Each DAO will be used to interact with a different table within the database schema.</p> 
	<p>The URI components will provide the front end a Restful entry point to request specific information. Each noun will have a respective UUL which can be used to collect and persist information. Below is a list of all the Restful URI endpoints</p>

<p>Rest URI Components</p>	<p>that are available for the application. For details on each please see the Restful API section below.</p>  <ul style="list-style-type: none"> · http://keyboard.cloudapp.net:3010/MusicService/user · http://keyboard.cloudapp.net:3010/MusicService/quiz · http://keyboard.cloudapp.net:3010/MusicService/demo · http://keyboard.cloudapp.net:3010/MusicService/question · http://keyboard.cloudapp.net:3010/MusicService/note · http://keyboard.cloudapp.net:3010/MusicService/tutorial · http://keyboard.cloudapp.net:3010/MusicService/tutorial_page · http://keyboard.cloudapp.net:3010/MusicService/tutorial_page?tutorialId=1 · http://keyboard.cloudapp.net:3010/MusicService/question?quizId=1 · http://keyboard.cloudapp.net:3010/MusicService/note?demoId=1
<p>Common</p>	<p>To ensure software reuse a folder called Common has been created to place common functionality under which does not fit under any of the above components. Some of the items that are common are the CorsFilter (Cross Origin Request Filter) which is used to allow the frontend access to make Restful requests. In addition a common logger utility was implemented to allow centralized logging of all service components.</p> 
<p>Test</p>	<p>For organization, all tests shall be stored under the test folder. This will allow Maven a central location to target to run all tests during a build process. As a good coding standard, each class shall contain the name of the noun that it is testing as well as the particular component of the noun it is testing.</p>

	
SQL	<p>As the Noteable application matures, it will require several database upgrades to be performed. To ensure sanity and limit the possibility of data being lost, all information shall be stored within a SQL folder. Each release of the database will have a separate version file. By having this standard it allows for developers to apply updates to their local database without overwriting any information that may have added locally. If a developer is creating the database for the first time they will run the latest create script. If the developer already has a version of the database but needs to update it, then they would run the alter scripts in chronological order from release n to release n+1 until they have their system up to date.</p> 

iv. Frontend Overview

The front end of the app consists of two html pages the content of which is updated by Angular and JQuery javascript code. The views associated with each page are as follows:

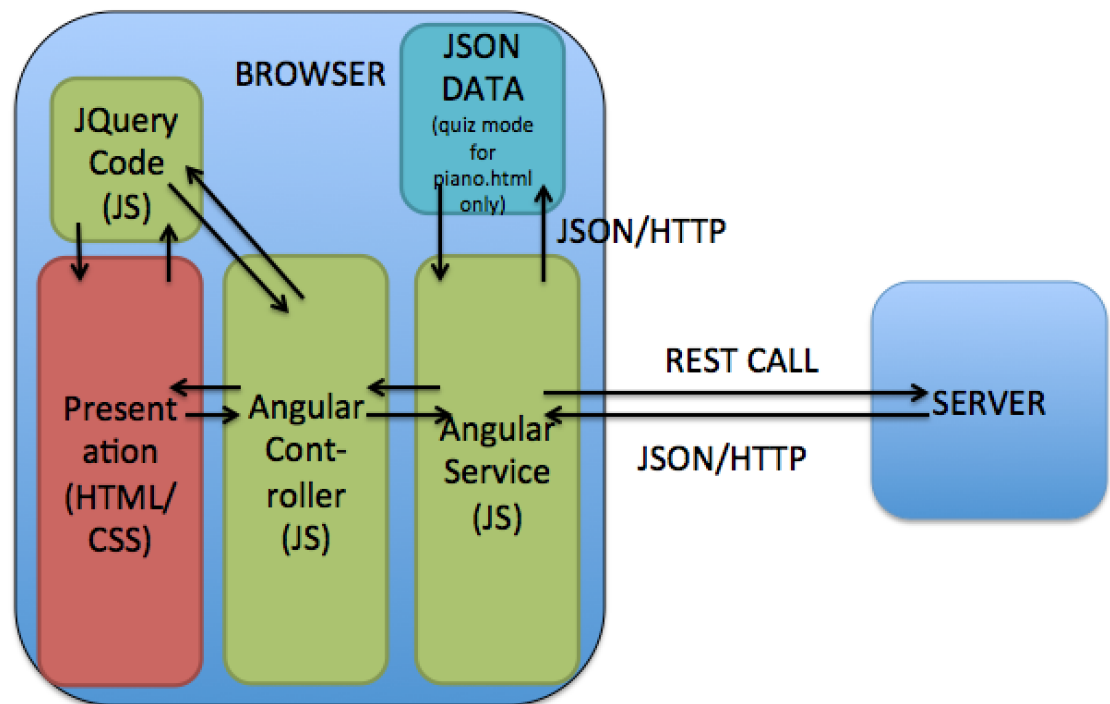
levels_selection.html:
tutorial menu,
quiz menu,
demo menu

piano.html
free play mode,
tutorial view,
quiz view,
demo view

The data defining the behavior of the app in each mode is obtained from the REST API.

v. Frontend Architecture

Both pages of the app have the same structure. Each page has it's own Angular.js module. Each module consists of an angular service, which is responsible for making the REST calls to the the back end api and formatting the data, and an angular controller which updates the html page in response to the received data. Both pages also have JQuery code which is called both by the HTML directly and by the angular controller.



Navigation between the two pages and between different page views is accomplished through links to custom routes. On route change, the angular controller for the page calls the appropriate function in the service to get the appropriate data for the view and updates the html. The following table shows all the views in the app, the route used to navigate to that view and the api call used to populate the data:

Level Selection page:

View	Route	API Call
Demo menu	level_selection.html#/demo	http://keyboard.cloudapp.net:3010/MusicService/demo
Tutorial menu	level_selection.html#/tutorial	http://keyboard.cloudapp.net:3010/MusicService/level
Quiz menu	level_selection.html#/quiz	http://keyboard.cloudapp.net:3010/MusicService/quiz

Piano Page:

View	Route	API Call
Free play mode	piano.html#/mode/free_play	none
Demo <demo_id>	piano.html#/mode/demo/level/<demo_id>	http://keyboard.cloudapp.net:3010/MusicService/note?demoId=<demo_id>
Quiz <quiz-id>	piano.html#/mode/quiz/level/<quiz_id>	http://keyboard.cloudapp.net:3010/MusicService/question?quizId=<quiz_id>
Tutorial <tutorial_id>	piano.html#/mode/tutorial/level/<tutorial_id>	http://keyboard.cloudapp.net:3010/MusicService/tutorial_page?tutorialId=<tutorial_id>

vi. Frontend Files and Major Functionality

level_selection.html:

File	Major Functionality
level_selection.html	HTML markup
Levels_Module.js	Angular module for levels page is initialized and routes are defined here
Levels_Service.js	Angular service <i>TutorialsNavService</i> , which makes api calls for the navigation pages is defined here.
Levels_Controller.js	Angular controller. Reacts if route changes. Calls appropriate function in <i>TutorialsNavService</i> and hands the data to <i>updateLevelsPage</i> to build the level selection page.
Level_Page_Scripts.js	JQuery scripts for level_selection.html. Main functionality is function <i>updateLevelsPage</i> which creates the the level selection menu table and adds it to the DOM.

piano.html:

File	Major Functionality
piano.html	HTML markup
Tutorial_and_Quiz_Module.js	Angular module for piano page is initialized and routes are defined here
Tutorial_And_Quiz_Services.js	Angular service <i>DataService</i> , which makes api calls to pull tutorial, quiz and demo data is defined here.
Tutorial_And_Quiz_Controllers.js	Angular controller. Reacts to route changes to display the requested quiz, demo or tutorial. Responsible for most of the DOM updates and user interactions during for quizzes, demos and tutorials.
Piano_Scripts.js	JQuery scripts for keyboard view. Maps

	piano keys and keys on the computer keyboard to sound objects to play a sound on click and stop playing it when the key is released.
--	--------------------------------------------------------------------------------------------------------------------------------------

Shared between piano.html and level_selection.html:

File	Description
Piano_Style.css	CSS styling for both pages

3. Design Patterns

For the backend, there were many design patterns used, many of which due to some of the best practices provided with Java Spring. Some examples of some of the patterns utilized are Singleton and the Template method.

One of the ways the **Singleton design pattern** will be implemented within the *Notable* application is via the logger object. Within the backend a logger object will be created at the start of the service. By creating one logger object and using it as a Singleton, it allows the ability to share the logger globally across the application; making it easier to centralize logs. This proved to be useful while debugging and verifying functionality was working as designed and completing tasks in the appropriate order.

The **Template method design pattern** provides the basic steps of a design within an abstract class. Later on subclasses can change the abstract functionality by overriding the abstract method. In spring this is used extensively for common code used to open and close connections. For examples are the JpaTemplate and JmsTemplate.

4. Key Algorithms

Within the NoteAble application there were no algorithms used. However, one of the algorithms that could be implemented in the future is the ability to track scores per quiz taken. This would be a very simple calculation to verify the score the user received. The score would then be verified to be at least in the 80th percentile to be

considered a passing grade for the quiz; which would be a prerequisite before being allowed to move onto the next quiz.

5. Classes and Methods

The current UML Class Diagram is shown below in figures 5-1 through 5-3. Due to the size of the UML Class diagram it has been divided into sections. The Object Aide Eclipse plugin was used to generate the class diagrams. Each object type was broken into three classes, the model class, the data access object (DAO) class, and the URI class. The model class represents the data model and utilization the hibernation annotation features maps the data objects to the specific DB table columns. The DAO class isolates the URI (REST API) class from the data model. The DAO class contains the methods for retrieving various collections of data such as `findById` or `findByParentId`. For example for the demo objects demo is the parent of notes and can be retrieved by the Demo ID or a specific Note ID. The URI class using Spring annotations maps the specific DAO method to a URI or URL. For example the Tutorial Page URI class maps the http://localhost:8080/MusicService/tutorial_page?tutorialId=id URI to the `findTutorialPagesByParentId` DAO method. Very similar structures were used for all three object types. An additional data object, `page_responses`, was added to the tutorial object type for handling the requirement of needing multiple expected key responses for a single tutorial page. the page response object has a many to one relationship with the tutorial page object.

a. Demo Objects

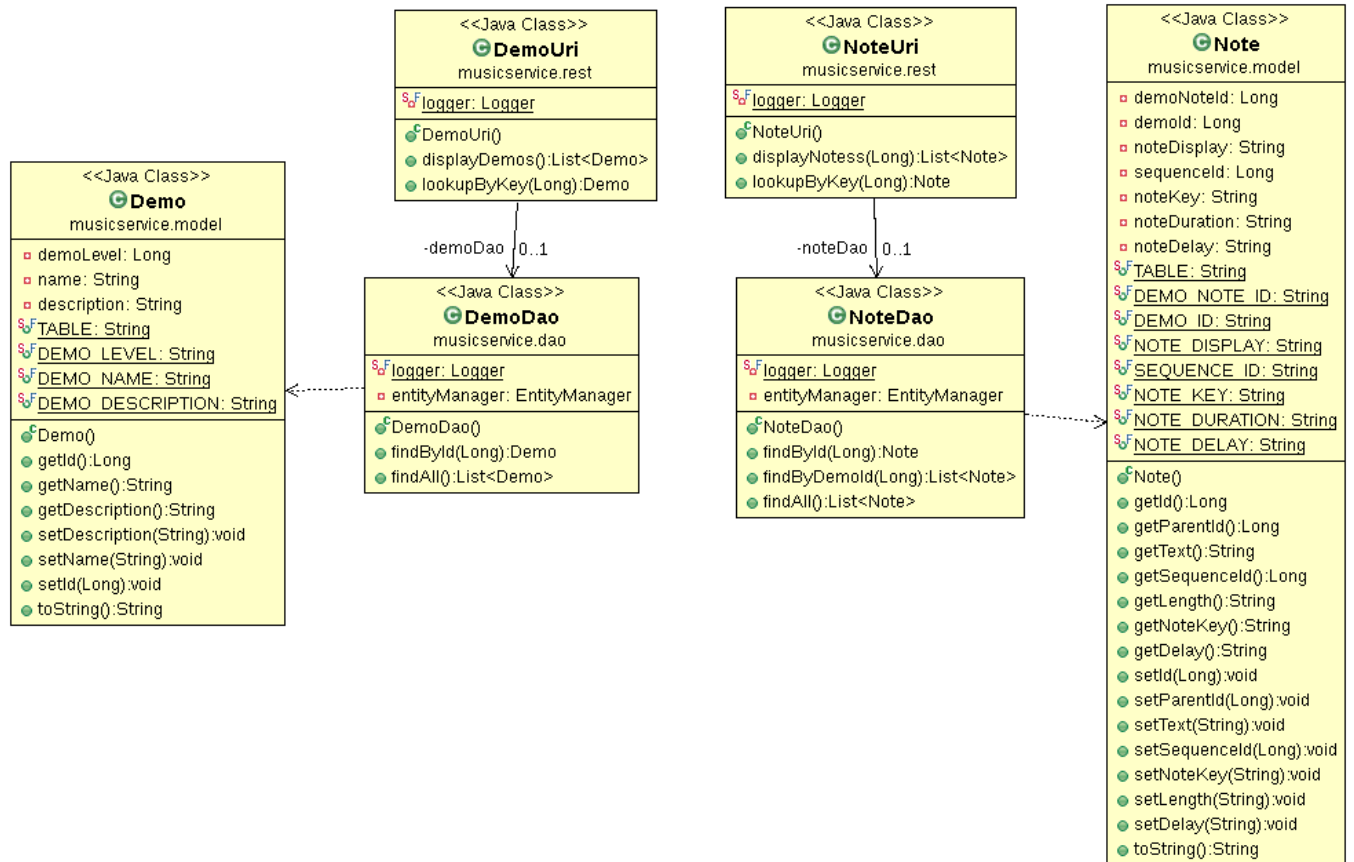


Figure 5-1

b. Quiz Objects

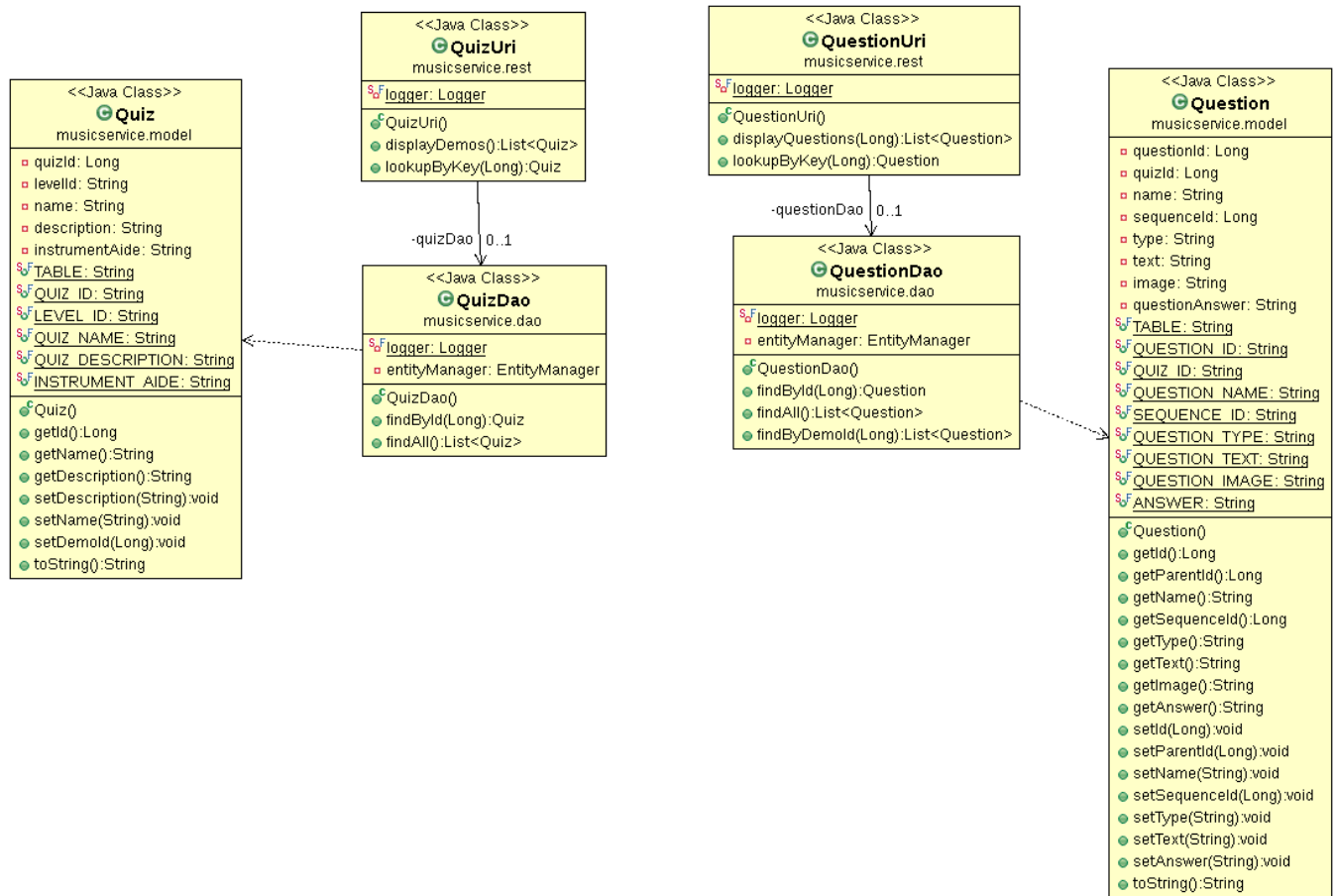


Figure 5-2

c. Tutorial Objects

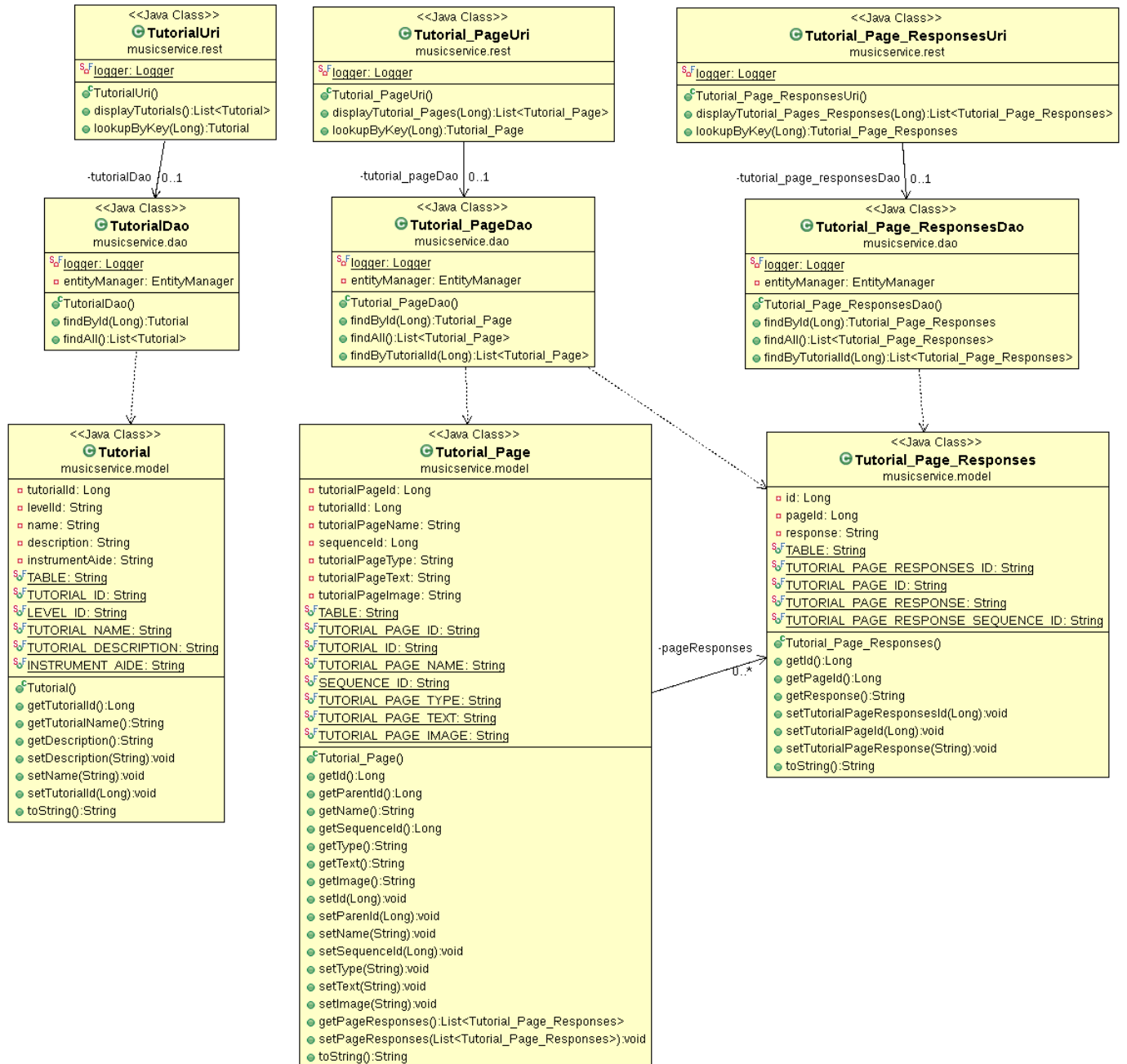


Figure 5-3

6. RESTful API

/user

This resource is used to manipulate user data. User data will contain information about each user of the application.

GET /user

Description:
This method will return information about all users in the database. Filters can be applied to limit the results. See inputs section for details about the filters that can be supplied.
Input:
username - The username of the user. role - Get the users which match the role supplied
Returns:
<pre>{ "user": [{ "-username": "pianoMan26", "-id": "1" "-role": "user" }, { "-username": "rockStar", "-id": "2" "-role": "admin" }, { "-username": "JamsterJenny", "-id": "3" "-role": "user" }] }</pre>

Example:

<http://localhost:8080/MusicService/user>

<http://localhost:8080/MusicService/user?username=pianoMan26>

POST /user

Description:

This method will insert a new user into the database. If a user already exists with the same email address, an error will be thrown.

Input:

```
{
  "username" : <username>,
  "role" : <role>,
  "email" : <email>,
  "first" : <first_name>,
  "last" : <last_name>
}
```

Returns:

id - The ID of the newly inserted user.

Example:

<http://localhost:8080/MusicService/user>

/quiz

This resource is used to manipulate quiz data.

GET /quiz/user/<id>

Description:

This method will return high level information about all quizzes for a specific user. It will provide information about the status of the user's progress per quiz. If a user has completed a quiz its status will be "Passed". If a user has started a quiz, its status will be "Started". If a tutorial has not been started yet, because the tutorial has not been completed, then the status will be "Locked".

Input:

None

Returns:

```
{
  "entries": {
    "quiz": [
      {
        "-name": "Quiz 1",
        "-id": "1"
        "-status": "Passed"
      },
      {
        "-name": "Quiz 2",
        "-id": "2"
        "-status": "Passed"
      },
      {
        "-name": "Quiz 3",
        "-id": "3"
        "-status": "Started"
      },
      {
        "-name": "Quiz 4",
        "-id": "4"
        "-status": "Locked"
      },
      {
        "-name": "Quiz 5",
        "-id": "5"
        "-status": "Locked"
      },
      {
        "-name": "Quiz 6",
        "-id": "6"
        "-status": "Locked"
      }
    ]
  }
}
```

}
Example:
http://localhost:8080/MusicService/tutorial/6

GET /quiz/<id>

Description:
This method will return details about a specific quiz level.
Input:
None
Returns:
<pre>{ "entries": { "question": [{ "-name": "Question 1", "-id": "1", "-question": "What hand do you use to play this note?", "-type": "Multiple_Choice", "-image": "http://localhost/images/quiz_1_q1.jpg", "-answer": "Right", "-":options": [{"-display": "Left"}, {"-display": "Right"}], }, { "-name": "Question 2", "-id": "1", "-question": "What hand is Shawn?", "-type": "Multiple_Choice", "-image": "http://localhost/images/quiz_1_q2.jpg", "-answer": "Right", "-":options": [{"-display": "Left"}, {"-display": "Right"}], }, { "-name": "Question 3",</pre>

```
    "-id": "1",
    "-question": "Which key is highlighted? ",
    "-type": "Multiple_Choice",
    "-image": "http://localhost/images/quiz_1_q2.jpg",
    "-answer": "A",
    "-":options": [
        {"-display": "A"},
        {"-display": "B"}
        {"-display": "C"}
        {"-display": "D"}
    ],
  ]
}
}
```

Tutorial Section

This sections includes the rest services that were used for the tutorial information

/tutorial_page?tutorialId=<id>

This method will return all tutorial pages for a certain level id.

GET /tutorial_page?tutorialId=<id>

Description:
This method will return high level information about all tutorial pages for a certain level. It will provide the ID, sequence ID, name and description of all available tutorials for a level.
Input:
None
Returns:
<pre>[{"pageResponses":[{"id":9,"pageId":16,"response":"f4"}],"sequenceId":1,"parentId":5,"image":"Tutorial_Level5_1.png","text":"Here is a new note, F! It is in the 1st space of the Treble Clef. This key is highlighted on the piano. Click the key or press the 'F' key on your piano.","id":16,"type":"press_key","name":"F Note"}, {"pageResponses":[{"id":10,"pageId":17,"response":"g4"}],"sequenceId":2,"parentId":5,"image":"Tutorial_Level5_2.png","text":"This is the note G. It is located on the second line of the piano. This note is also played with your right hand. Click the note on the piano or press 'G' to hear what it sounds like!","id":17,"type":"press_key","name":"G Note"}, {"pageResponses":[],"sequenceId":3,"parentId":5,"image":"Tutorial_Level5_3.png","text":"Here we start with two f notes. The arrows show when the notes change. All of these notes will be played with your right hand on a real piano.","id":18,"type":"press_continue","name":"Two F Notes and Two G Notes"}, {"pageResponses":[],"sequenceId":4,"parentId":5,"image":null,"text":"Congratulations you have finished the 5th Tutorial! Click continue to move on to the quiz.","id":34,"type":"done","name":"Conclusion"}]</pre>

Example:

http://localhost:8080/MusicService/tutorial_page?tutorialId=5

/tutorial_page/<id>

This method will return all information for a specific tutorial page.

GET /tutorial_page/<id>

Description:

This method will return all information for a specific tutorial page. It will provide the ID, parent ID, sequence ID, name, image location, type and description.

Input:

None

Returns:

```
{ "pageResponses": [{ "id": 1, "pageId": 12, "response": "c4" }, { "id": 2, "pageId": 12, "response": "c4" }, { "id": 3, "pageId": 12, "response": "c4" }, { "id": 4, "pageId": 12, "response": "c4" }, { "id": 5, "pageId": 12, "response": "c4" }, { "id": 6, "pageId": 12, "response": "c4" } ], "sequenceId": 3, "parentId": 3, "image": "Tutorial_Level3_3.png", "text": "Here there are 6 middle C notes in a row. The first three should be played with the right hand, and the last three should be played with the left hand on a real piano.", "id": 12, "type": "press_key", "name": "Six Middle C Notes" }
```

Example:

http://localhost:8080/MusicService/tutorial_page/12

/tutorial_page_response/<id>

This method will return all information for a specific tutorial page response.

GET /tutorial_page_resopnse/<id>

Description:
This method will return all information for a specific tutorial page response. It will provide the ID, parent ID, and response.
Input:
None
Returns:
<pre>{"id":12,"pageId":20,"response":"a4"}</pre>
Example:
http://localhost:8080/MusicService/tutorial_page_response/20

/tutorial

This method will return all high level information about tutorials

GET /tutorial

Description:
This method will return all high level information about tutorials
Input:
None
Returns:

```
[{"tutorialId":1,"description":"This is the first tutorial and is a great place to start.","tutorialName":"Level 1 Tutorial"},  
  
{"tutorialId":2,"description":"This is the second tutorial and should not be started until the first tutorial is completed..","tutorialName":"Level 2 Tutorial"},  
  
{"tutorialId":3,"description":"This is the third tutorial and should not be started until the second tutorial is completed..","tutorialName":"Level 3 Tutorial"},  
  
{"tutorialId":4,"description":"This is the fourth tutorial and should not be started until the third tutorial is completed..","tutorialName":"Level 4 Tutorial"},  
  
{"tutorialId":5,"description":"This is the fifth tutorial and should not be started until the fourth tutorial is completed..","tutorialName":"Level 5 Tutorial"},  
  
{"tutorialId":6,"description":"This is the sixth tutorial and should not be started until the fifth tutorial is completed..","tutorialName":"Level 6 Tutorial"},  
  
{"tutorialId":7,"description":"This is the seventh tutorial and should not be started until the sixth tutorial is completed..","tutorialName":"Level 7 Tutorial"},  
  
{"tutorialId":8,"description":"This is the eighth tutorial and should not be started until the seventh tutorial is completed..","tutorialName":"Level 8 Tutorial"}]
```

Example 1:

Example:

<http://localhost:8080/MusicService/tutorial>

7. References

8. Glossary