

Persisting Data in Relational Databases

Introduction

Previous assignments described using HTML, React and Redux to build a dynamic single page application (SPA) as a prototype of a course manager application called WhiteBoard. Web services exposed a server side data model that stored application state on the server. This assignment expands on the Web services and focuses on persisting the server model to an equivalent relational model implemented in a MySQL database server. JPA (Java Persistence API) will be used to map the Java server data model to the relational model.

For this assignment you will create or refactor Java classes that model widgets. JPA will be used to map the Java classes to database tables that will store records for widget instances.

Learning objectives

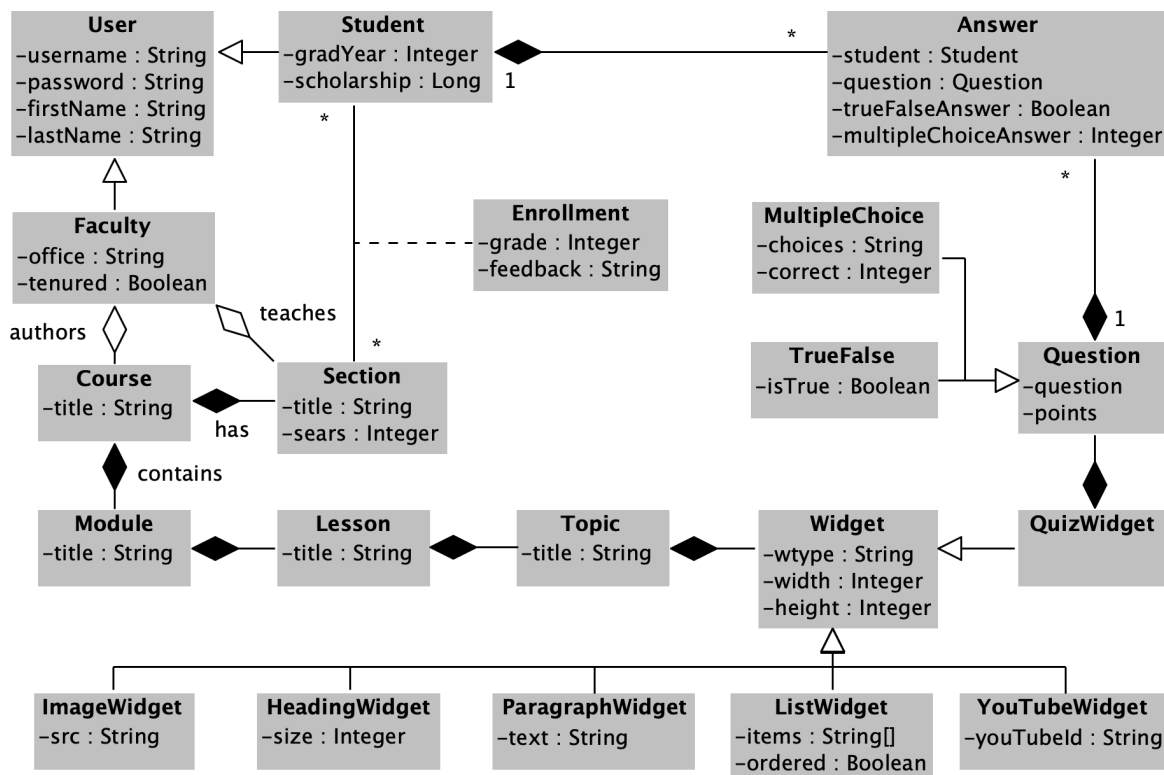
The learning objectives of this assignment are

- Mapping Java data models to relational models
- Integrating Redux with RESTful Web services

Features

The features implement it in this assignment are

- Persist widgets for a given topic
- Retrieve, post, update, and delete widgets from a REST Web service



Create java data models

Based on the model shown above, create class ***models/Widget.java***. Implement a JPA entity mapped to a SQL table named "widgets". The entity should contain the following properties

Class Variable	Type	Description
id	Integer	Unique identifier for this widget
topicId	String	Topic this lesson belongs to
name	String	Optional name of the widget
type	String	Type of the widget, e.g., Heading, List, Paragraph, Image, YouTube, HTML, Link
widgetOrder	Integer	Order with respect to widgets in the same list
text	String	Plain text useful for heading text, paragraph text, link text, etc
src, url, and/or href	String	Absolute or relative URL referring to online resource
size	Integer	Useful to represent size of widget, e.g., heading size
width, height	Integer	Widget's horizontal & vertical size, e.g., Image's or YouTube's width & height
cssClass	String	CSS class implementing some CSS rule and transformations configured in some CSS rule
style	String	CSS transformations applied to the widget
value	String	Some arbitrary initial value interpreted by the widget

Create repositories

In ***repositories/WidgetRepository.java*** implement a JPA CRUD repository for ***Widget***. Repositories provide create, read, update, and delete operations (CRUD) that will allow manipulating data stored in the database.

Create web services

In ***services/WidgetService.java***, implement (or refactor) a service class that manipulates a list of widget instances stored in the database. Remove all use of the array implementation and replace the implementation to use the WidgetRepository. Refactor ***controllers/WidgetController.java*** to use the new WidgetService implementation

Method	Description
--------	-------------

Widget createWidget(Integer tid, Widget widget)	Creates a new Widget instance and saves it to a widgets table for a topic whose ID is tid. Returns new widget inserted in the database
List<Widget> findWidgetsForTopic(Integer tid)	Returns collection of all widgets for a topic whose ID is tid
Widget updateWidget(Integer wid, Widget widget)	Updates widget whose id is wid encoded as JSON in HTTP body. Returns 1 if successful, 0 otherwise
void deleteWidget(Integer wid)	Removes widget whose id is wid. Returns 1 if successful, 0 otherwise
List<Widget> findAllWidgets()	Returns collection of all widgets
Widget findWidgetById(wid)	Returns a single widget instance whose id is equal to wid

Implement widgets

Implement widgets **ListWidget** and **ImageWidget**. Create or refactor necessary components, reducers, and services.

Implement List widget

In **src/components/widgets/list-widget.js**, implement component **ListWidget** that users can use to create lists of items. The list widget provides a textarea element in which users can enter each list item in a separate row. The text is used to generate an HTML list as shown here. Clicking the **edit icon** (cog) displays the widget in edit mode revealing an Ordered checkbox and "List Items" textarea.

Lists are unordered by default, e.g., Ordered checkbox is unchecked. Users can check the checkbox to change the list type between ordered and unordered. If the list type is ordered, then the list will render with `` and if it is unordered, then it will render with ``. The items in the list are captured as a simple plain text that is then split into multiple list items. The placeholder should be *Enter one list item per line*. Clicking the **save icon** (checkmark), saves the changes to the server, hides the form elements and renders the list based on the saved configuration. Clicking on the **delete icon** (trashcan) removes the widget permanently from the server and re-renders the list of widgets without the deleted widget.

Implement image widget

In **src/components/widgets/image-widget.js** implement component **ImageWidget**. Images can be

added with the image widget where users can provide a URL to an image as shown here. Clicking the ***edit icon*** (cog) displays the widget in edit mode and reveals text input fields for configuring how the widget should render. The ***Image URL*** widget property captures the URL of the image used to render the image. The placeholder should be ***Image URL***. The ***Image width*** and ***height*** properties capture the size of the widget.







Image URL

Image width

Image height

Deliverables

As a deliverable, check in all your work into a github source control repository. Deploy your project to a remote public server on Heroku or AWS. Submit the URL of all repositories and remote servers on blackboard. Tag the commit you want graded as assignment6. TAs will clone your repository and grade the tagged commit.