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

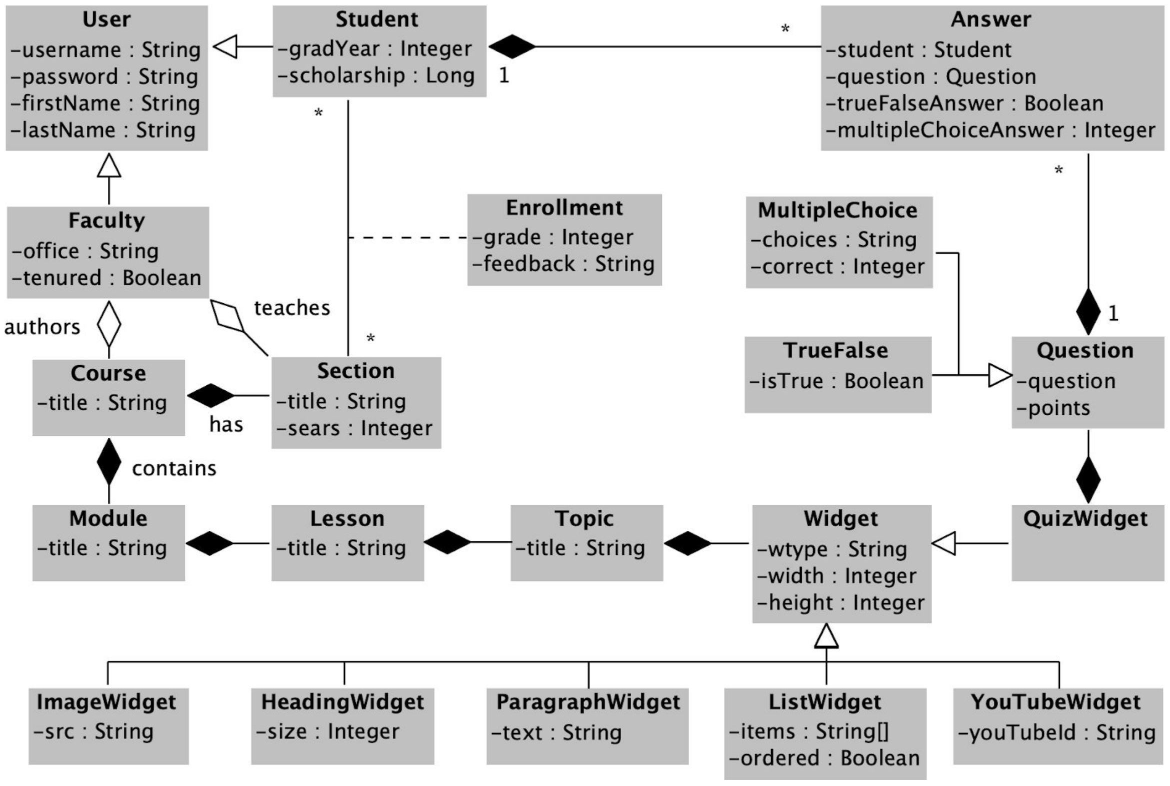
The learning objectives of this assignment are

- 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 (20pts)

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 (10pts)

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. Feel free to work on additional repositories.

Create web services (20pts)

In **services/WidgetService.java**, implement (or refactor) a service class that manipulates a list of widget instances. A RESTful controller will expose the service as a Web service mapping the API of a service to a set of HTTP requests.

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

Create controllers (20pts)

In **controllers/WidgetController.java**, implement a controller that provides the following RESTful endpoints mapped to the corresponding handler methods:

HTTP	URL pattern	Method	Description
POST	**/api/topics/{tid}/widgets**	**Widget createWidget (int tid, Widget widget)**	parses Widget JSON object from HTTP body encoded as JSON string. Uses WidgetService to create a new Widget instance and add it to the existing collection of widgets for a topic whose ID is tid. Returns the new widget with a unique identifier
GET	**/api/topics/{tid}/widgets**	**List<Widget> findWidgetsForTopic(int tid)**	uses WidgetService to retrieve collection of all widgets and returns a string encoded as a JSON array for a topic whose ID is tid
PUT	**/api/widgets/{wid}**	**int updateWidget(int wid, Widget widget)**	parses Widget JSON object from HTTP body encoded as JSON string. Uses WidgetService to find widget instance whose id is equal to wid and update the fields to be the new values in widget parameter. Returns 1 if successful, 0 otherwise.
DELETE	**/api/widgets/{wid}**	**int deleteWidget(int wid)**	uses WidgetService to remove widget whose id is wid. Returns 1 if successful, 0 otherwise.

GET	**/api/widgets**
	List<Widget> findAllWidgets()
	uses WidgetService to retrieve collection of all widgets and returns a string encoded as a JSON array.
GET	**/api/widgets/{wid}**
	Widget findWidgetById(int wid)
	uses WidgetService to retrieve a single widget instance whose id is equal to wid and returns a string encoded as a JSON object.

Create JavaScript web service clients

Create JavaScript Web service clients for each of the data models exposed through RESTful Web services. The Web service client will provide modular data access to a React application to interact with a remote Web service API. Some of the services might have been implemented in previous assignments. Refactor existing services and create new services as needed.

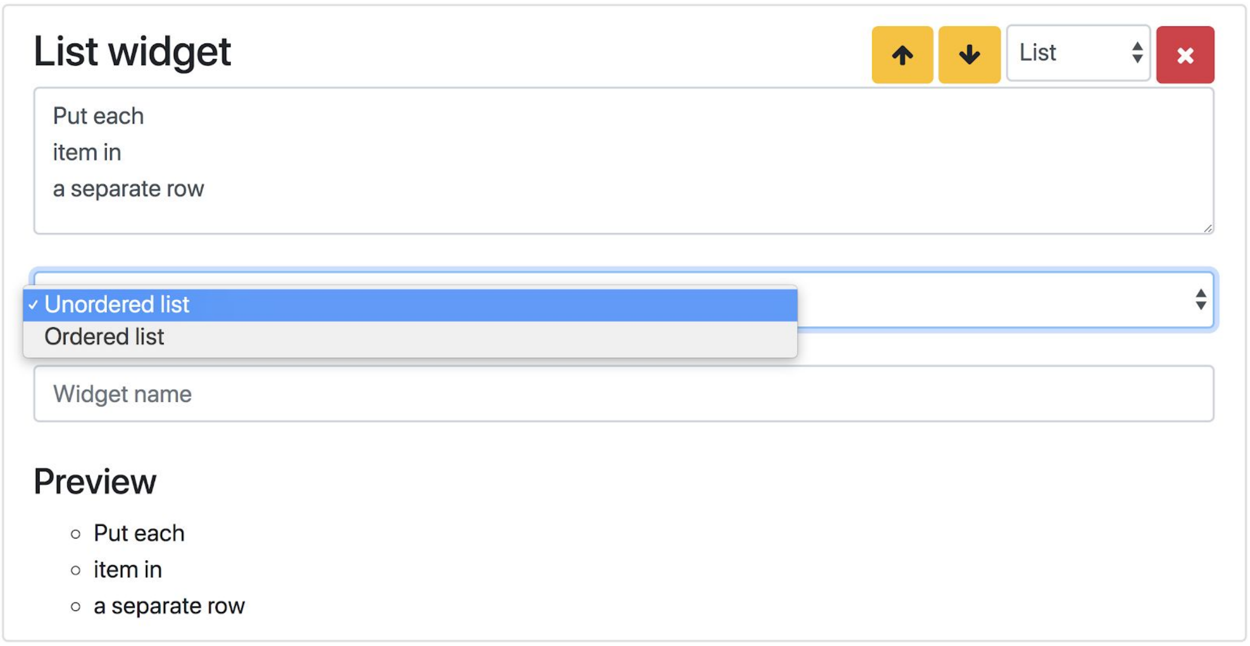
Note that depending how you designed the rest of the client application, you might not need to use these services. Implement and test them nevertheless. Unused services should at least provide the operations create, find all, find by id, update, and delete.

Implement widgets (20pts)

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

Implement List widget (10pts)

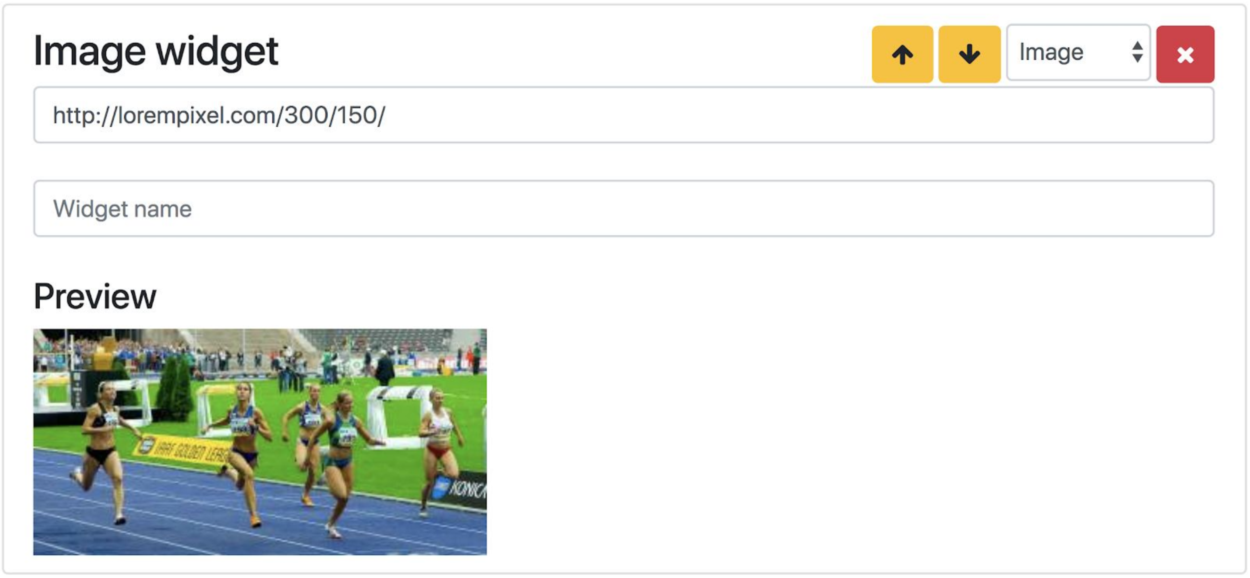
Implement component **src/components/WidgetListComponent.js**. Users can create lists with the list widget. The list widget provides a textarea element in which users can enter each list item in a separate row. The text is used to generate an HTML list as shown below.



Lists are unordered by default. Selecting the list type from the list type dropdown changes the type to either unordered or ordered. The preview of the list is re-rendered as the list type is changed or the list items is changed in the textarea. 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*.

Implement image widget (10pts)

Implement component **src/components/ImageWidget.js**. Images can be added with the image widget where users can provide a URL to an image as shown below. The *image URL* widget property captures the URL of the image used to render the image. The placeholder should be *Image URL*.



Conventions, naming, file name and directory names (5pts)

Client side conventions

- ######Component
  - Stateless React JS components should end with Component
  - Implement stateless React JS components as ES6 arrow functions
  - Stateful React JS components which use the state for internal layout and behavior should end with Component
  - All components should be in a folder called components
- ######Container
  - Stateful React JS components which provide state and event handlers for childd elements should end with Container
  - All container components should be in a folder called containers
- ######Service
  - All communication between client and server should be implemented in separate service files
  - The name of service files should end with Service
  - All services should be in a folder called services

Server side conventions

- ######Controller
  - All controllers should be in a folder called controllers
  - Defines URLs to expose a data model as web services
  - Defines POST end point to create data
  - Defines GET end point to retrieve data
  - Defines PUT end point to update data
  - Defines DELETE end point to remove data
  - URLs contain plural nouns of data being manipulated
  - Uses services to manipulate data model
  - Singleton
- ######Service
  - All services should be in a folder called services
  - Declares CRUD operations to manipulate data model
  - Agnostic to protocol
  - Singleton

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.