RESTful Web Services

Accessing Server Data through RESTful Web Services

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. The SPA was implemented entirely as a client side application running entirely on a browser. This assignment expands on SPA to provide server side support and state. Feel free to reuse the HTML, CSS, React and Redux implemented used in previous assignments. The SPA will interact with RESTful services implemented using Spring Boot.

Learning objectives

The learning objectives of this assignment are
- Implement RESTful Web services
- Implement a data model using Java
- Expose a data model through a REST API
- Retrieve data from REST Web services
- Post data to REST Web services
- Update and delete data from REST Web services

Create a Spring Boot Java server application

If you have not done so already, create a spring boot java server application using the spring command line as follows:

spring init whiteboard --dependencies=web

Import the maven project generated into your favorite IDE

Implement Widget RESTful service

As the user interacts with the user interface (UI) on the browser, the React.js application modifies the DOM to reflect the changes. Changes in the client are temporary and can only be made permanent with the help of a server. Clients can notify servers of changes in the state of the application through the use of HTTP requests to specific URL endpoints published by the server through a technology called AJAX (Asynchronous JavaScript and XML). We refer to these endpoints as Web services. RESTful Web services impose certain conventions on the syntax and methods of the HTTP URLs used to communicate with the Web services. The conventions are based on best practices applicable when the end points are used to manipulate some data model on the server. Data models usually provide create, read, update, and delete operations (CRUD) operations. RESTful Web service conventions state that URLs should be nouns describing the data model, e.g., /widgets, /courses, /modules, etc. REST conventions also state that HTTP POST will be used to create,

DELETE to remove, GET to read, and PUT to update instances of the data model. Create a RESTful Web service to persist the state of the application.
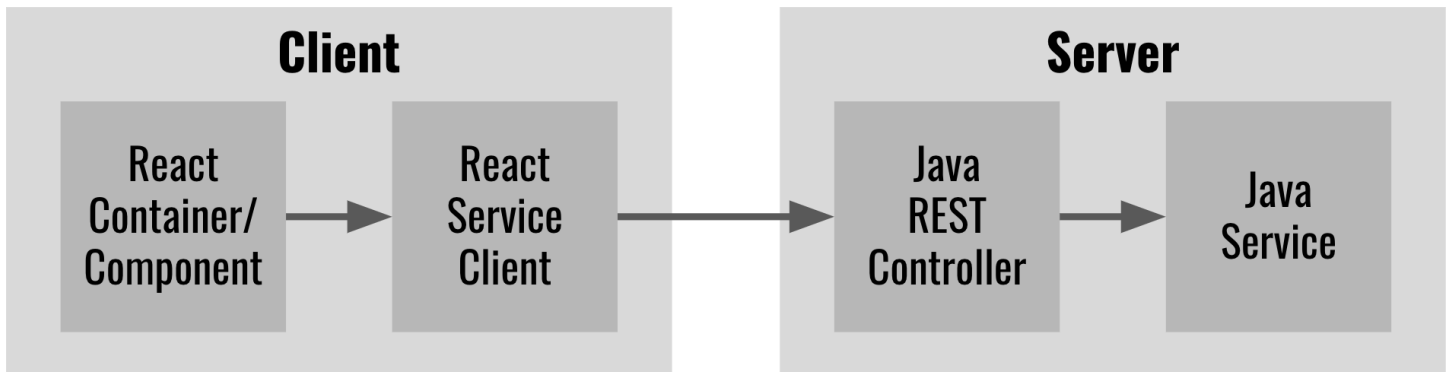
Implement Widget data model

In **models/Widget.java**, implement a data type that models the concept of user interface widgets described earlier. Group under one Java class all the attributes necessary to represent the various types of widgets. Below is a suggested schema for **Widget.java**. Feel free to add additional fields if needed

Class Variable	Type	Description
name	String	Optional name of the widget
id	Integer	Widget's unique identifier
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

Implement service WidgetService.java

In **services/WidgetService.java**, implement a service class that manipulates a list of widget instances. Maintain the list of widgets as a Java collection stored in a local class variable. Later assignments will persist the data to a database. A RestfulController will expose the service as a Web service mapping the API of a service to a set of HTTP requests.

Method	Description
Widget createWidget(String tid, Widget widget)	Creates a new Widget instance and add it to the existing collection of widgets for a topic whose ID is tid. Returns new widget with a unique identifier
List<Widget> findWidgetsForTopic(String tid)	Returns collection of all widgets for a topic whose ID is tid
int updateWidget(String wid, Widget widget)	Updates widget whose id is wid encoded as JSON in HTTP body. Returns 1 if successful, 0 otherwise
int deleteWidget(String wid)	Removes widget whose id is wid. Returns 1 if successful, 0 otherwise
List<Widget> findAllWidgets() (optional)	Returns collection of all widgets (optional)
Widget findWidgetById(wid) (optional)	Returns a single widget instance whose id is equal to wid (optional)

Implement RESTful Web service WidgetController.java

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

HTTP	URL pattern	Method		Description
POST	**/api/topics/{tid}/widgets**			
	Widget createWidget (String 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(String 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(String 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(String wid)**			

uses WidgetService to remove widget whose id is wid. Returns 1 if successful, 0 otherwise.

GET   **/api/widgets (optional)**

         List<Widget> findAllWidgets()

uses WidgetService to retrieve collection of all widgets and returns a string encoded as a JSON array.

GET   **/api/widgets/{wid} (optional)**

         Widget findWidgetById(String wid)

uses WidgetService to retrieve a single widget instance whose id is equal to wid and returns a string encoded as a JSON object.

Implement RESTful Web service Client widget-service.js

In your React.js project, create a new **services/widget-service.js** and integrate it to the new **WidgetController.java** Web service to delete, find, update, and create widgets on the client and store the state on the server. Refactor any React containers, components, and reducers accordingly to use the new implementation of WidgetService.js to integrate with the remote Java server.

Method	Description
createWidget(tid, widget)	**POST**s to **/api/topics/{tid}/widgets** a JSON object encoded as a string representing a new widget instance
findWidgetsForTopic(tid)	**GET**s all widgets as a JSON array from **/api/topics/{tid}/widgets** for a topic whose ID is tid
findAllWidgets() (optional)	**GET**s all widgets as a JSON array from **/api/widgets**
findWidgetById(wid) (optional)	**GET**s a single widget as a JSON object from **/api/widgets/{wid}**
updateWidget(wid, widget)	**PUT**s to **/api/widgets/{wid}** a JSON object encoded as a string representing an existing widget instance with new values for the attributes for a widget whose ID is wid
deleteWidget(wid)	**DELETE**s an existing widget whose ID is wid from **/api/widgets/{wid}**

Widget List

+

In **/components/widgets/widget-list.js**, create a react component called **WidgetList** that renders an array of widgets for the currently selected topic. A different list should display as the user selects a different topic. The component must be connected to a reducer that provides its state and handles its events. Below is an example of the widget list component rendering the heading and list widgets. There are different types of widgets: Heading, Paragraph, List, Image, Hyperlink, and Video. **Only implement the Paragraph and Heading** widgets for this assignment.

On the right is an example of the widget list component rendering the heading and paragraph widgets. Clicking the *add widget button* (plus sign on the top right) adds a new widget at the bottom of the widget list. Default widget type is Heading of size 1.

Each widget has an edit button (cog) that toggles it into edit mode. Clicking on the save button (checkmark) saves the widget updates to the server and toggles it back to its preview mode hiding all the form elements and showing the edit button again. Clicking on the edit button hides the edit button and reveals all the other form elements, e.g., *widget type dropdown*, delete, save, titles, inputs, textareas, preview, etc. clicking on save updates the widget on the server, hides the form elements, toggles the widget back to preview mode, and shows the edit button again.

While in edit mode, clicking the *delete widget button* (trashcan) permanently removes the widget from the server and the list re-renders without the removed widget. Refreshing the page before the widget list is saved can lose all the changes. Clicking the *save widget button* (checkmark) saves the widget to the server and toggles the widget out of edit mode.

While in edit mode, each widget has a *widget type dropdown* shown above. The widget's type can be changed at any time by selecting the widget's widget type from the *widget type dropdown*. Changing the widget's type re-renders the widget to match the new widget type with the forms remembering the values the user might have already entered.

Heading not being edited (default) ⚙

| Heading | ⌄ | 🗑 ✓ |

Heading being edited

| Heading 1 | ⌄ |

Paragraph not being edited (default). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ⚙

| Paragraph | ⌄ | 🗑 ✓ |

Paragraph being edited. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu

```
✓ Heading
  Paragraph
  Video
  Image
  Link
  List
  HTML
```
widget type dropdown

```
✓ Heading 1
  Heading 2
  Heading 3
  Heading 4
  Heading 5
  Heading 6
```
heading size

Paragraph Widget

In **/components/widgets/paragraph-widget.js**, implement component **ParagraphWidget** as shown below. ParagraphWidgets render plain text surrounded by a paragraph element <p>. Clicking the edit button (cog) toggles the widget into edit mode. In edit mode, the edit button is hidden and the paragraph is replaced by a textarea element for editing the paragraph. Users can use the widget type dropdown to change the type of the widget. The delete button (trashcan) permanently removes the widget from the server and the widget list. The save button (checkmark) saves the paragraph to the server, exist edit mode, and reveals the paragraph again with the new updates.

Paragraph not being edited (default). Lorem ipsum dolor sit amet, consectetur adipiscing ⚙ elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Paragraph	⌄	🗑 ✓

Paragraph being edited. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu

Heading Widget

In **/components/widgets/heading-widget.js**, implement component **HeadingWidget** as shown below. HeadingWidgets render plain text surrounded by a heading element <h1>, <h2>, ..., or <h6>. Clicking the edit button (cog) toggles the widget into edit mode. In edit mode, the edit button is hidden and the heading is replaced by an input element for editing the text of the heading. Users can use the heading size dropdown to configure the size of the heading. Use the widget type dropdown to change the type of the widget. The delete button (trashcan) permanently removes the widget from the server and the widget list. The save button (checkmark) saves the heading to the server, exist edit mode, and reveals the heading again with the new updates.

Heading not being edited (default) ⚙

Heading ⌄

Heading being edited

Heading 1 ⌄

🗑 ✓

Widget Reducer

In **src/reducers/widget-reducer.js** create a Redux reducer that implements the action types shown below. Add additional actions as needed.

Action Types	Description
CREATE_WIDGET	creates a new widget instance for the topic whose ID is topicId
DELETE_WIDGET	removes an existing widget
UPDATE_WIDGET	updates an existing widget
FIND_ALL_WIDGETS_FOR_TOPIC	retrieves all widgets for a particular topic
FIND_ALL_WIDGETS (optional)	retrieves all widgets
FIND_WIDGET (optional)	retrieves a particular widget

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 in blackboard. Tag the commit you want graded as assignment5. TAs will clone your repository and grade the tagged commit.

This assignment updates two repositories and two Heroku remote servers. Submit a link to all relevant Git repositories and heroku remote servers.