Team DEC13-15

Task Manager

Final Document

Dalia Abo Sheasha

Cameron Legleiter

Martin Strobel

TABLE OF CONTENTS

THIS DOCUMENT

The scope of this document is to summarize our (DEC13-15, xmlers) work on the Task Manager project for Shashi Gadia. This summary will include details on implementation accomplishments and testing results.

In addition, this document will include a description of how to install and interact with our product.

FINAL DESIGN

Please see our Design Document for an up-to-date account of our design and implementation strategies.

IMPLEMENTATION AND TEST RESULTS

IMPLEMENTATION

We have successfully created a working edition of our product, which posterity should find easy to maintain. In this section, we will detail the sections of the code that are implemented.

FRONT-END

Here we discuss the end-user facing functionality that was completed, and any conclusions we've drawn along the way.

CROSS-BROWSER SUPPORT

An issue that tends to plague web designers is creating web pages that can be used between multiple types of browsers with little visual difference. This issue was also initially prevalent when designing pages for Task Manager. What made development easier was using the CSS framework Bootstrap by Twitter. Bootstrap offers a basic layout and consistent look and feel to a website that does not require the designer/developer to have to know too much or write very specific CSS for specific parts of webpages. Using Bootstrap allowed us to make simple, easy to follow webpages with a consistent look to each page. Additionally, JavaScript libraries such as Modernizr extend functionality to older browser versions, in particular Internet Explorer 8 and older.

REUSABLE HTML TEMPLATES

With the create page, a major problem came up right away with creating multiple form entries. Since a form may contain multiple fields of the same type, it became imperative to find a way to reuse multiple parts of the HTML. By utilizing an MVC (Model View Controller) framework combination of Backbone.js and Marionette.js, it was trivial to create a single HTML template view to represent one form entry, and reuse the template in the case multiple entries are created. This framework can later be extended to handle more complex entry types; since complex types are merely made up of simple types, all of which can be combined into, again, a reusable template, it will come down to combining existing templates to create complex ones. However, the major tradeoff with these frameworks is a steep learning curve, meaning developers who may continue implementing new features with Task Manager must spend time both learning how to use the framework, and understanding the code currently written.

BACK-END

Here we will discuss the portion of the code that was developed that will be installed on a server, and interacted with very little except by those people hoping to modify our website.

DATA CONVERSION

Among the first issues we discussed upon starting this project was how we would pass data around, and which modules would accept which types of data. It quickly became clear that we needed a robust and easy means of converting Form data between several different types, all of which may be added to or removed in the future.

Because the nature of forms is to have multiple types of permissible fields, we had to choose a pattern that allowed us to evaluate lists of non-homogenous data. We could have accomplished this using Java's built-in polymorphism. However, we decided early that we could make our code considerably more maintainable by using the visitor pattern, which would separate an object from the details of how it is transformed. Most notably, it would allow for multiple implementations of transformations that may otherwise have confusingly similar names. The best example that we ended up implementing was the difference between generating JSON data that included response data from a form, versus generating a form to be filled out.

For more information on the Visitor Pattern, see: http://www.oodesign.com/visitor-pattern.html

Ultimately, using the Visitor Pattern was a successful choice. There were, however, stressful moments when we ran-up against problems with our understanding of the pattern, and against language limitations on Java's part.

DATABASE IMPLEMENTATION

Our advisor was very clear that the requirement he most desired was that our implementation emphasize XML. Knowing this, it would have been relatively easy to merely use a technology similar to Saxon, which would allow us to directly communicate between XML and the Servlets.

However, we believe that quality software is easily changed, and not entirely dependent on a single technology.

It was with this philosophy in mind that we developed a layer of abstraction allowing us to use any database implementation we chose. The implementations that we wrote included: XML, JSON, and a debugging-only implementation.

TESTING

UNIT TESTING

We were able to use unit testing effectively for approximately 90% of our back-end programming. The parts where good coverage was not achieved were interactions based on web-requests.

MANUAL TESTING

For all holes in automated testing, and general development, we relied on manual testing. This largely comprised of spawning a web-server on the developer's laptop.

PERFORMANCE TESTING

Unfortunately, hardware and time constraints limited the amount of performance testing we were able to complete. In particular, we host the site locally on our laptops for development, so any performance metrics would be next to meaningless. However, it is unlikely that we would see a significant reduction in performance. We speculate that in the environment this site is likely to be launched, performance will be adequate.

OPERATIONAL MANUAL

Basic interactions with the website should be intuitive, but for sake of completeness will be repeated here. In addition, steps to run the site will be included here.

INSTALLATION

PREREQUISITES

Some form of webserver that can process Java Servlets should be installed on the administrator's machine. During development, Apache Tomcat was used. More information on this webserver can be found here: http://tomcat.apache.org/

If one wishes to use the JSON strategy for a database controller, MongoDB must be installed on the administrator's machine. More information on this webserver can be found here:

http://www.mongodb.org/

If any other database strategy is employed, there is no other prerequisites for installation.

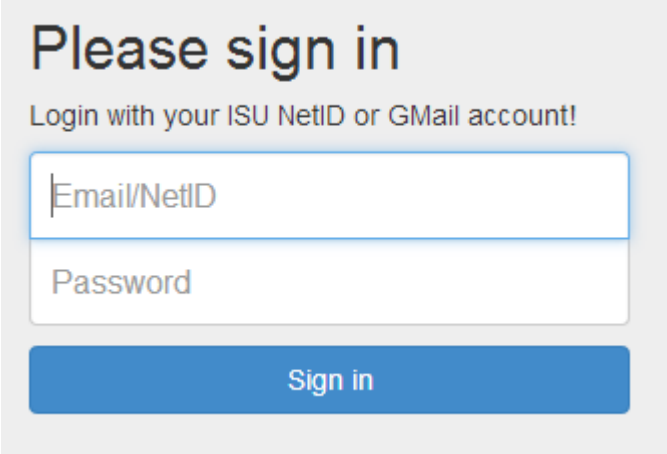
ENVIRONMENT VARIABLES

If no environment variables are set, defaults will be used. However, system administrators may be uncomfortable with the defaults. For best results, set all of the environment variables described below.

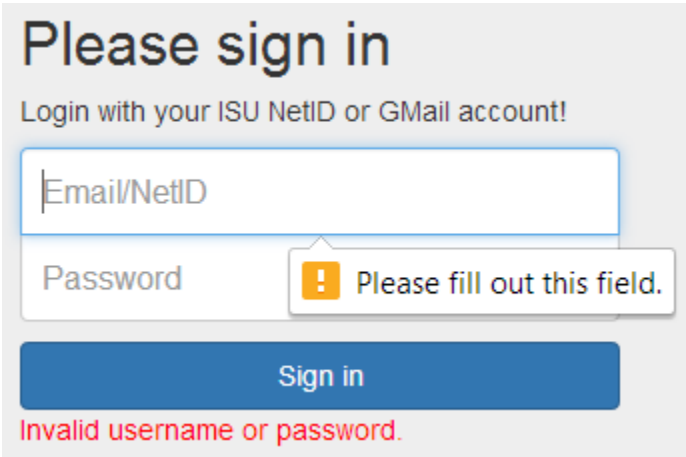
Variable Name	Valid Values	Description
XMLERS_STRATEGY	"XML", "JSON", "DEBUG"	Selects which database controller will be used
XMLERS_FORM_LOCATION	Any XML file-path	The XML file that contains form data, or where that file should be located.
XMLERS_USER_LOCATION	Any XML file-path	The XML file that contains user data, or where that file should be located.

INTERACTION

1. Login Page

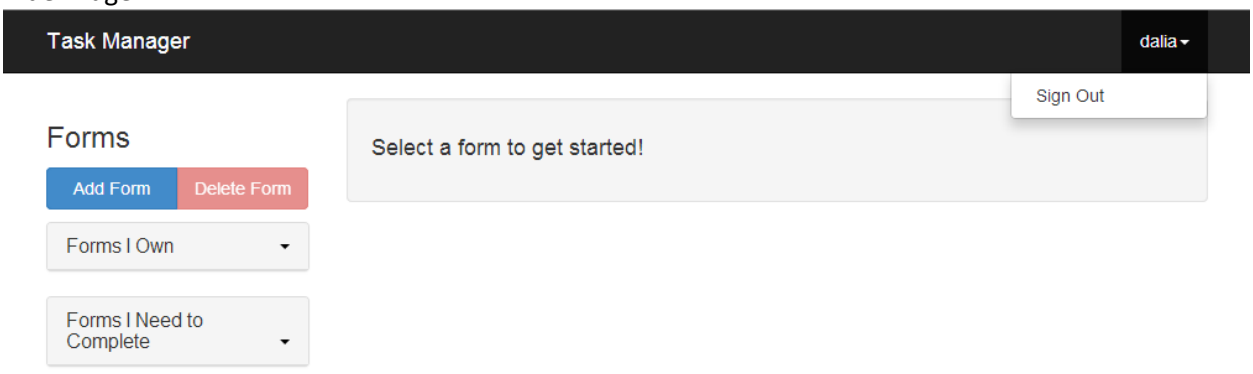


Users login to our web application using their IASTATE username and passwords. Since we are using GoogleService for authentication, a user may login with their GMAIL as well. If a user wants to login with their IASTATE account, they are not required to add in the "@iastate.edu" portion of their email, only the Net-ID. However, if users are logging in with GMAIL, they must add the "@gmail.com" portion when signing in.



If a user enters in login information incorrectly, a message will be displayed to alert them. Also, if they fail to fill in either fields, they will be alerted as well.

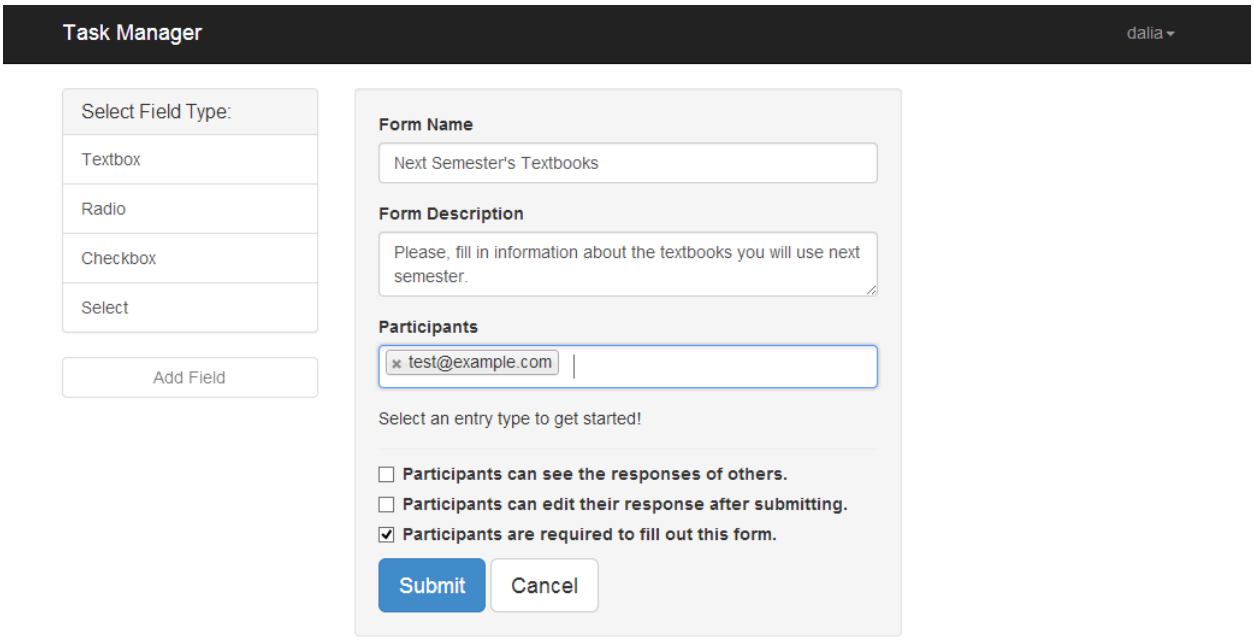
2. Index Page



Once a user logs in, they will be directed to their homepage where:

- Create a new form.
- View forms the user is an owner of.
- Delete forms the user is an owner of.
- View forms the user is a participant.
- Sign out of the user's account.

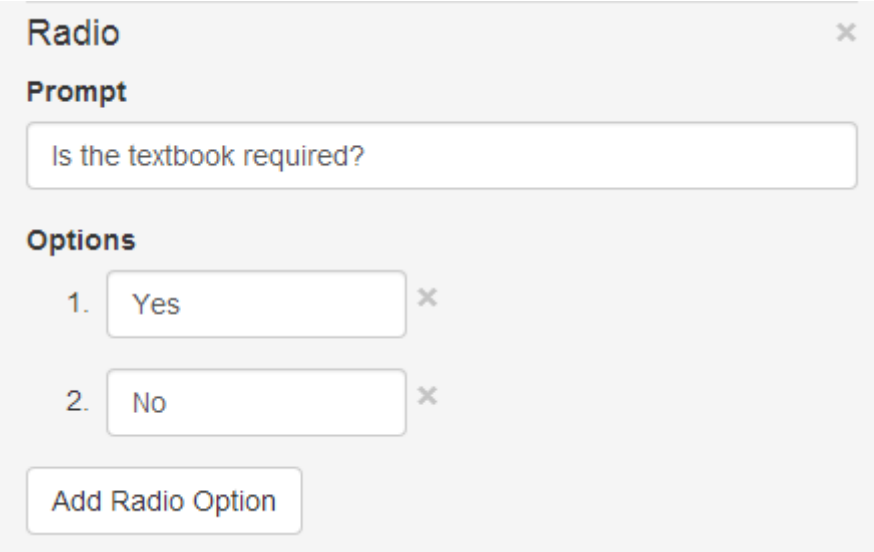
3. Create Form



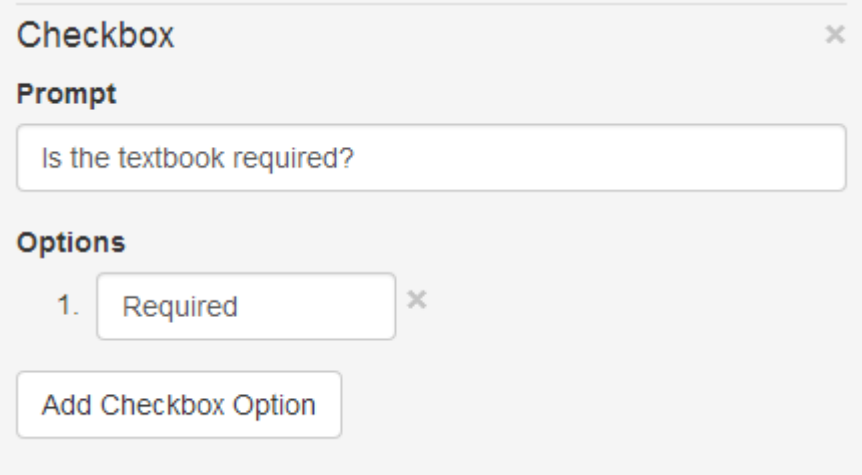
a. Once a user clicks on "Add Form", they will be prompted to specify:
   - The form's name.
   - A description of the form.
   - The participant's email addresses delimited by commas or space. When entering email addresses, make sure to press enter or the space bar in order for the application to recognize the email.
   - A flag that allows other participants to view the form's records. Otherwise, other participants are restricted and are not allowed to access the form's data.
   - A flag that allows participants to go back to the form and resubmit their responses.
   - A flag that allows participants to unparticipate from a form.

b. Then users are expected to add fields based on the type they require.

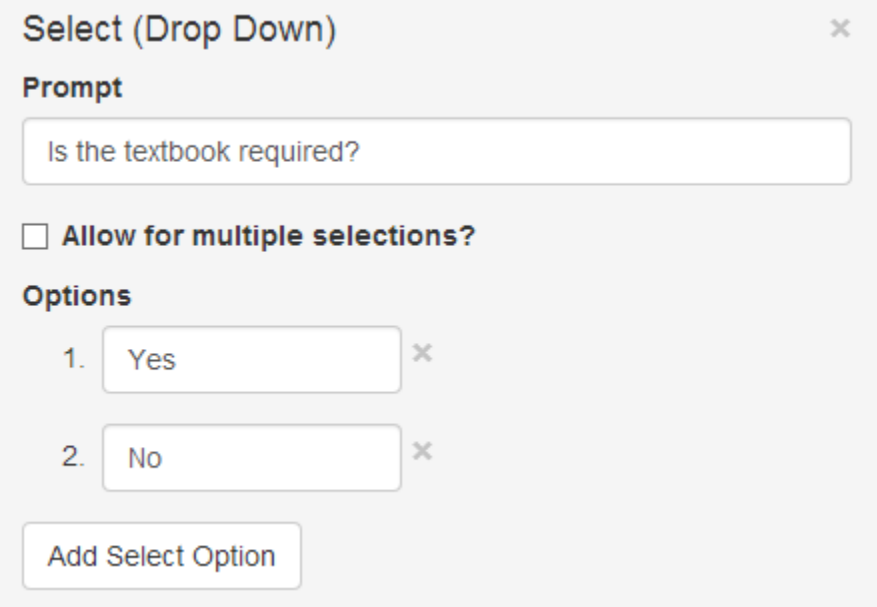
- Textbox type. The user fills in:
  - The prompt the participants will be presented with to fill in their response.
  - The maximum number of characters for the response's field.



- Radio type. The user fills in:
  - The prompt the participants will be presented with to fill in their response.
  - The options for the radio buttons the participants will see. They will be able to add as many options as they please.



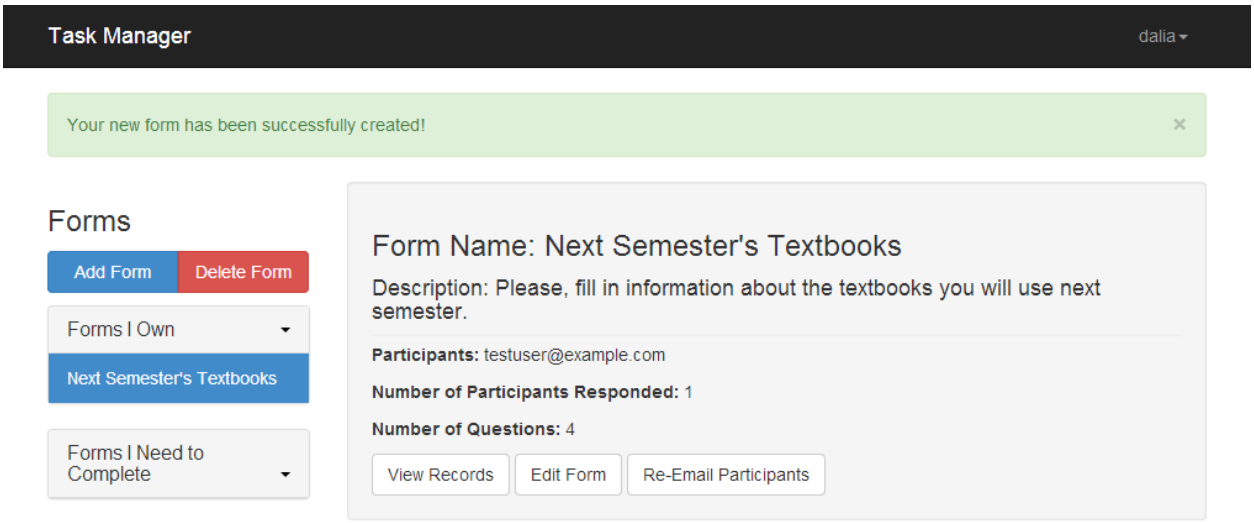
- Checkbox type. The user fills in:
  - The prompt the participants will be presented with to fill in their response.
  - The options for the checkboxes the participants will see. They will be able to add as many checkboxes as they please.



- Select type. The user specifies:
  - The prompt the participants will be presented with to fill in their response.
  - A flag for whether the select drop down menu will allow users to select multiple options.
  - The options for the select drop down menu the participants will select from. They will be able to add as many checkboxes as they please.

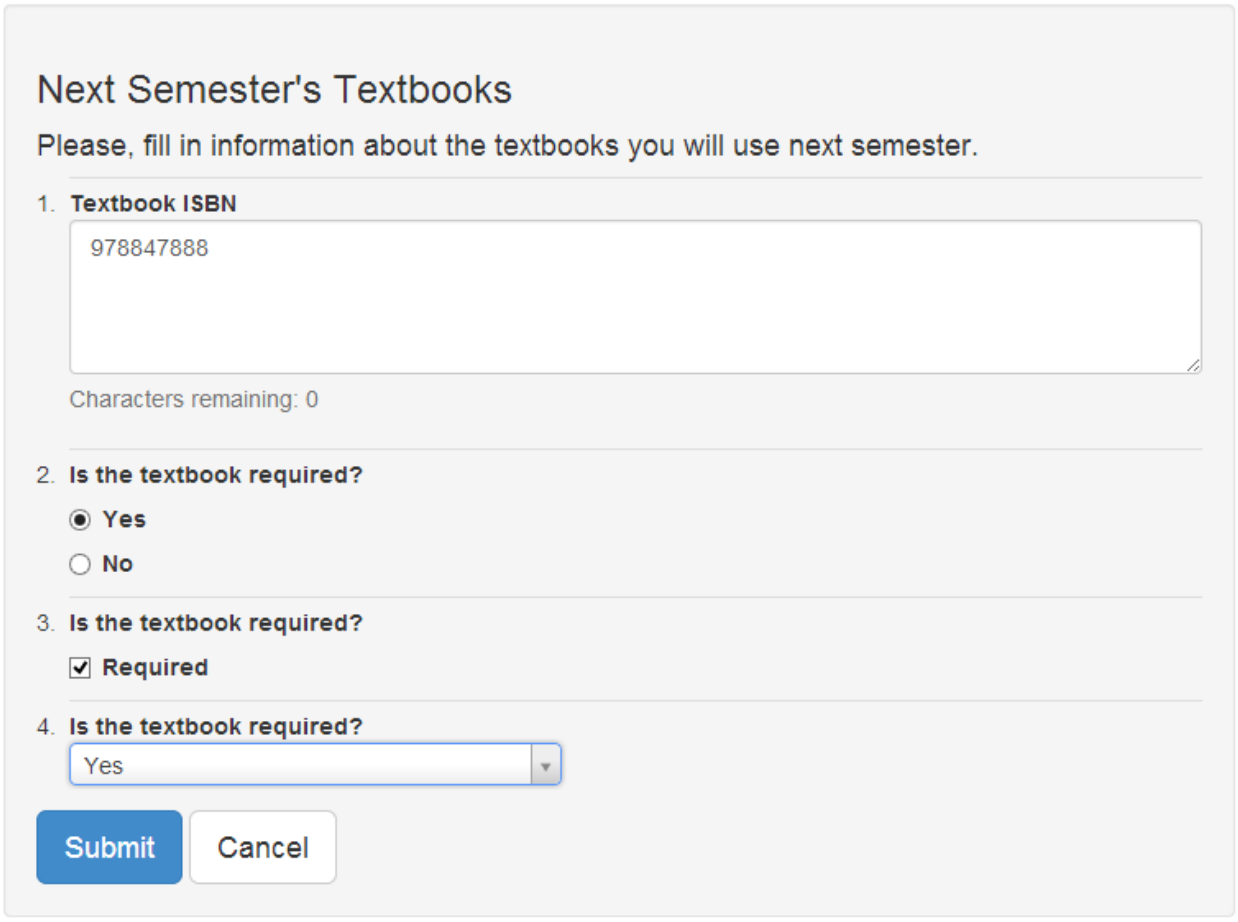
c. User may choose to submit the form or cancel out. If a user submits a form, an email will be sent to the participants informing them that their response is expected.

4. View Form.



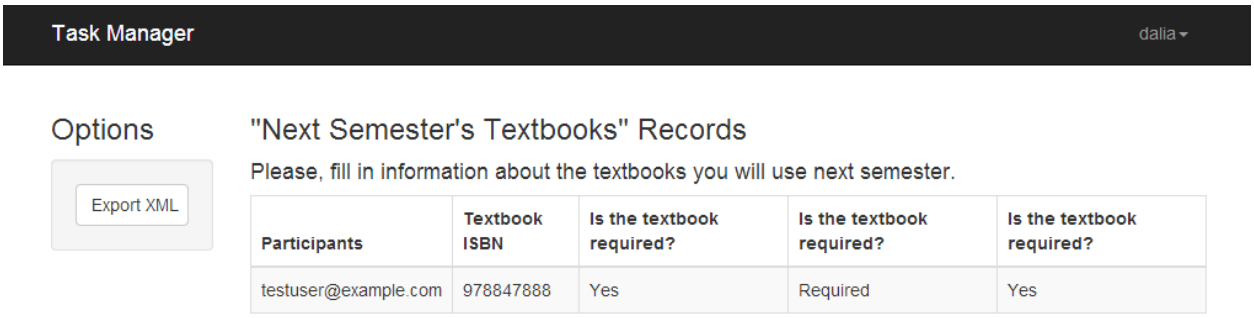
A user can then go back to their homepage to view the new form's information. There, they can access the responses of participants, edit the form, or re-email participants that haven't responded yet.

5. Respond to a form.



- A user is able to submit a response corresponding to the fields of the form they are a participant of.

6. View Form Records.



The owner of the form can view the participant's responses. Owners view the data in table form where they can organize the data in the table by each of the columns. They are also given the option to export the data to XML. When a user chooses the export option, an XML document is downloaded to the owner's machine.