MSIMBO

Database Development

**Description**

This assignment adds database persistent storage to the ongoing class assignment. Previous assignments have used client side and server side data management. This assignment uses mongoose data models and schemas to persist and interact with a MongoDB instance. Web services will be refactored to use the data models instead of previous mock data collections. Communication with the database will need to be asynchronous using promises to register callbacks from the database.

**Connect to MongoDB Database**

In command port:

Mongod – installs mongo

Mongo – launches mongo shell

1. Install mongoose

**> $ npm install mongoose –save**

**Note:** Install in the Assignment folder.

1. Create an empty model folder in server folder

**Implement a Data Model**

Use mongoose to implement schemas and models for all the assignment's entities: users, websites, pages, and widgets. Refactor Web services to use the mongoose models instead of the static mock data used in previous assignments. Mongoose models must return promises to interact asynchronously with MongoDB. Web services must handle promises returned by the mongoose models. Implement schemas and models under a directory called .**/server/model**. Make sure to apply good programming practices discussed in class such as declaring module APIs towards the top of the module and implementation functions towards the bottom of the module, format source code with white spaces to improve readability and follow naming conventions. Group schemas and models under a directory for the specific entity. Here's a list of all the schemas and models required for this assignment

**./server/model/models.server.js – Will connect with Mongo database**

**./server/model/user/user.schema.server.js**

**./server/model/user/user.model.server.js**

**./server/model/website/website.schema.server.js**

**./server/model/website/website.model.server.js**

**./server/model/page/page.schema.server.js**

**./server/model/page/page.model.server.js**

**./server/model/widget/widget.schema.server.js**

**./server/model/widget/widget.model.server.js**

Set up /server/mondel/models.server.js for localhost and Heroku

|  |
| --- |
| var connectionString = 'mongodb://127.0.0.1:27017/web-dev'; // for local  if(process.env.MLAB\_USERNAME\_WEBDEV) { // check if running remotely    var username = process.env.MLAB\_USERNAME\_WEBDEV; // get from environment    var password = process.env.MLAB\_PASSWORD\_WEBDEV;    connectionString = 'mongodb://' + username + ':' + password;    connectionString += '@ds0000.mlab.com:29004/dasdasds'; // use yours  }  var mongoose = require("mongoose");  var db = mongoose.connect(connectionString);  module.exports = db; |

Implement a User Data Model

Implement a Mongoose User Schema

Implement a Mongoose schema with the following properties and data types

|  |  |  |
| --- | --- | --- |
| **Property** | **Data Type** |  |
| username | String |  |
| password | String |  |
| firstName | String |  |
| lastName | String |  |
| email | String |  |
| dateCreated | Date | Current date |

Implement a Mongoose User Model

|  |  |
| --- | --- |
| **Function Signature** | **Description** |
| createUser(user) | Creates a new **user** instance |
| findUserById(uid) | Retrieves a user instance whose **\_id** is equal to parameter **uid** |
| findUserByUsername(username) | Retrieves a user instance whose **username** is equal to parameter **username** |
| findUserByCreadentials(username, password) | Retrieves a user instance whose **username** and **password** are equal to parameters **userId** and **password** |
| updateUser(uid, user) | Updates user instance whose **\_id** is equal to parameter **uid** |
| deleteUser(uid) | Removes user instance whose **\_id** is equal to parameter**uid** |

Implement a Website Data Model

Implement a Mongoose Website Schema

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Data Type** | **Default Value / Description** |
| developerId | Reference to **User** | Refers to parent user |
| name | String |  |
| description | String |  |
| dateCreated | Date | Current date |

Implement a Mongoose Website Model

|  |  |
| --- | --- |
| **Function Signature** | **Description** |
| createWebsiteForUser(website) | Creates a new **website** instance |
| findAllWebsitesForUser(uid) | Retrieves all website instances for user whose  **\_id** is **uid** |
| findWebsiteById(wid) | Retrieves single website instance whose **\_id** is **wid** |
| updateWebsite(wid, website) | Updates website instance whose **\_id** is **wid** |
| deleteWebsite(wid) | Removes website instance whose **\_id** is **wid** |

Implement a Page Data Model

Implement a Mongoose Page Schema

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Data Type** | **Default Value / Description** |
| \_website | Reference to Website | Refers to parent website |
| name | String |  |
| description | String |  |
| dateCreated | Date | Current date |

Implement a Mongoose Page Model

|  |  |
| --- | --- |
| **Function Signature** |  |
| createPage(page) |  |
| findAllPagesForWebsite(wid) |  |
| findPageById(pid) |  |
| updatePage(pid, page) |  |
| deletePage(pid) |  |

Implement a Widget Data Model

Implement a Mongoose Widget Schema

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Data Type** | **Default Value / Valid Values** |
| \_page | Reference to Page | Refers to parent page |
| type | String, enum | ['HEADING', 'IMAGE', 'YOUTUBE', 'HTML', 'INPUT'] |
| name | String |  |
| text | String |  |
| placeholder | String |  |
| description | String |  |
| url | String |  |
| width | String |  |
| height | String |  |
| rows | Number |  |
| size | Number |  |
| class | String |  |
| icon | String |  |
| deletable | Boolean |  |
| formatted | Boolean |  |
| dateCreated | Date | Current date |

Implement a Mongoose Widget Model

|  |  |
| --- | --- |
| **Function Signature** |  |
| createWidget(pageId, widget) | Creates new **widget** instance for parent page whose **\_id** is **pageId** |
| findAllWidgetsForPage(pageId) | Retrieves all widgets for parent page whose **\_id** is **pageId** |
| findWidgetById(widgetId) | Retrieves widget whose **\_id** is **widgetId** |
| updateWidget(widgetId, widget) | Updates widget whose **\_id** is **widgetId** |
| deleteWidget(widgetId) | Removes widget whose **\_id** is **widgetId** |
| reorderWidget(pageId, start, end) | Modifies the order of widget at position **start** into final position **end** in page whose **\_id** is **pageId** |

**Deliverables**

GitHub and Heroku Deliverables

To allow instructor to see your changes, please frequently commit and push your work to GitHub and OpenShift repositories. Below is an example of the commands you will use. The example assumes your project is located in **~/MSIMBO/web-dev**:

**> cd ~/MSIMBO/web-dev**

**> git add .**

**> git commit -m 'A comment describing your work'**

**> git push github**

If using Heroku, you might need to deploy from within the Heroku dashboard. If you configured Heroku to auto deploy when the repo was updated, then you should be all set.

Verify that the files have copied to the github repository. Also visit your Heroku website and verify that your changes are reflected on the remote server.

Tagging a Release

When you consider your work complete and ready for evaluation (ready for release), go to your code repository in GitHub and generate a release by navigating to "releases". Then click on "Create a new release" and type the name of the tag in the input field labeled "Tag version". We will be using the following tags for the various assignments:

assignment4 (previous assignment)

**assignment6 (this assignment)**

If you need to resubmit the assignment then create a new tag by adding a version number, e.g.,

assignment5.1, assignment5.2, etc...

I will grade the very last release. The date/time you create the tag will be considered the date/time of submission.