Booksterdam Log

**11 November 2016 (5hrs)**

* Was able to manipulate the database hosted in azure using SQL Server Management Studio.
* Wrote queries to create database schema, and some insertions and selections
* **ISSUES**:
* INT data type can’t hold more than 10ish characters. So, that’s a problem for storing the 13 digit ISBN. Use BIGINT. **(DONE)**
* Also, there are two ISBNs for books. So, need to add an additional ISBN -13 field.
* Make ISBN Not Null in ShelvesRecords as well, since there is no way to display book details if the user doesn’t provide ISBN. All we will be able to see is the BookID, seller info and price but no details of the book. **(DONE)**
* **TASK**: prepare further queries. Assignable to anyone else if interested. Just give them the description of queries that will be needed and they can write it for you.

**12 November 2016 (6hrs)**

* Added script to setup the connection to remote database, handler to basic query requests, and displaying of retrieved results in the web page.
* Changed datatype of ISBN in BookRecords and ShelvesRecords to BIGINT.
* Also made ISBN in ShelvesRecords NOT NULL.
* Added state NOT NULL column to CustomerRecords
* **TASK**: Need to develop a primitive front end layout and handle basic requests to the database to see how well it goes. (done partially with retrieval of customerRecords)
* **ISSUE**: The users page right now doesn’t retrieve updated results from the database until the server is restarted.

**13 November 2016 (2hrs)**

* Tried using module.exports to export the db connection, but didn’t work due to unpredictable asynchronous function executions. Had to revert to previous commit.

**17 November 2016 (4hrs)**

* Got help from sujil in setting up a global database connection. It can make requests now.
* Need to learn how to properly implement callback functions to wait updating displays until data is received.
* Implement those callbacks in index.js and/or dbRequester.js

**19 November 2016 (6hrs) FIRST DEBUG DEPLOYMENT**

* Implemented call back function successfully. Basically this is how it works:
* DB function

Function RequestMaker (callback) {

Make requests to database and put it into ‘data’ object

Callback(data); //This is like a function that has to be later overloaded by the caller

}

* Caller

RequestMaker(function (data) {

Console.log(data);

});

* **FIRST DEPLOYMENT IN DEBUG MODE**
* There are two master branches for the local repository from now on: **origin master** and **azure master**
* The deployment needed an azure installed as a node package
* Update in the database was reflected properly when the website was refreshed. So, the communication seems to be established well.
* Made a basic layout by introducing Boostrap into the design. Used the boostrap examples on the websites named Grid and Jumbotron.

**20 November 2016 (9hrs)**

* Added the SchoolRecords table. Also added school column (NOT NULL) to ShelvesRecords as it would simplify refining items based on school. The change is integrated into the create table SQL queries.
* Maybe school column should be added to CustomerRecords later but we’re letting that go for now
* Made thumbnail layouts for book displays using Bootstrap and also refined the existing table displays.
* **CHOICE OF API:** IsbnDB seems to be the best API as it also displays the price details from various vendors along with other details we need and summaries (No Images). Google API provides preview pages of books which is cool and images too. Amazon seems to be more sales geared and also provides less info on query. OpenLibrary.org has an easy way to retrieve images but it’s database isn’t extensive. So, maybe we’ll go with isbnDB for book’s info and Google for the images. IsbnDB can’t be dropped easily as it gets the rates.
* Also added url column to the BookRecords table.
* All the necessary information is retrieved and displayed in thumbnail at this point.
* The app is capable of displaying all the books in the records successfully as thumbnails.
* **LESSON:** Had issues with uneven height in columns, and the matchHeight plugin helped me solve it.

**25 November 2016 (15mins)**

* Added the require statement for the Database.js in app.js hoping that it will allow the connection to be set initially when the server is started. Previously the connection was done when the index page was loaded. Seemed to be working fine locally.

**17 December 2016 (15mins)**

* Exported the connection object instead of the request object from database.js. Created request object inside dbRequests.js while making a request so that the if a request is made it can pick a fresh available connection from the pool.
* **Update**: This seems to have solved the issue of failing to make db requests after few hours of server startup. Website has been up and running for almost a day without any errors while loading.

**18 December 2016 (4hours)**

* Added the search bar and experimented with freezing it at top. **Not a good idea** to freeze.
* Experimented with displaying two columns of items on phone**. Not a good idea.**
* Experimented with some color contrasts and found something decent for now.
* **FONT CHANGE:** Just need to include it in layout.js and drop it in custom css.
* **ISSUE:** The search button doesn’t seem to be functioning because it doesn’t change at hover.

**19 December 2016 (6hours)**

* **Query**
* Added a query to display matching results based on what the user enters in search bar.
* The query searches matches in order of ISBN, title, author, publisher, school. Can add more.
* Query has been parameterized to handle sql injection. Tested with selections.
* Search results are quite satisfactory.
* **Ajax**
* Implemented Ajax call to successfully display search results. (Refer to viewmanager.js for reference)
* Divided webpage to multiple jade templates to facilitate easy update of selected view based on Ajax requests.

**Milestone**

* As of 12/19/2016 (before visiting Nepal), the web application successfully displays all the books on the database and allows users to search available books.