Intro

Frameworks come and go. Databases, however, tend to stick around. My solution for this system would be database focused. All business logic should be pushed to the database. The frontend should then simply serve as a wrapper for interacting with these objects and serving up their results. In other words, I would fall back to an assignment 3 approach. Specifically, I would use straight PHP and MySQL. PHP would be used on the frontend. MySQL would be used as the backend database. A web server running Apache Tomcat would then be used to host the site. In the following sections, I will discuss the following topics in greater detail:

- Business Requirements
- Assumptions
- Entity Model / Database
- MVC Elements and their relationships
- Major Visual layout considerations
- Security and Access Control
- Deployment

Business Requirements

At the outset the following business level requirements were provided by the organization:

- *A community library needs an inventory management system.*
- *They have several kinds of inventory items including:*
  - *Books*
  - *Blu-ray disks*
  - *magazine*

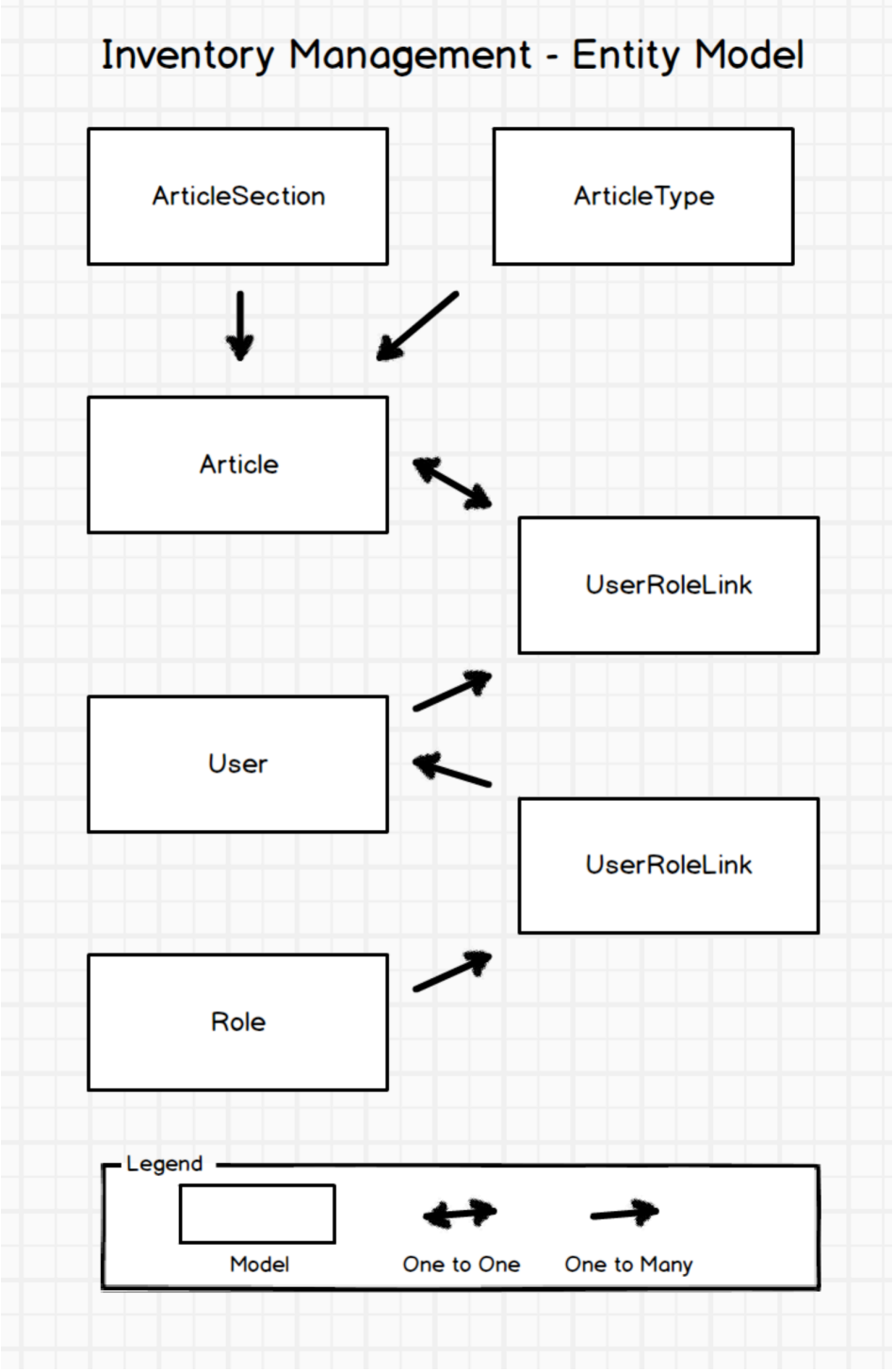
  NOTE - the library can have several copies of each.

- *The library manager needs to be able to–*
  - *Add new inventory items*
  - *mark old ones not for loan when they are retired.*
- *Library desk operators need to be able to*
  - *scan inventory barcodes*
  - *record the loan date*
  - *record due date*
  - *record the person who loaned the item.*
- *For books the library wants to–*
  - *keep track of which Authors are the most popular*
  - *have a function to determine how many Books from a given Author are on loan.*
- *Books, magazines and disks are all organized into library sections (for a given articles dewy decimal number)*
- *Managers should be able to–*
  - *manage library sections, viewing the total number of available books in each section.*
- *Finally users of the library should be able to–*
  - *log in to the system*
  - *view their loaned items, ideally*
    - *sorted by –*
      - *type*
      - *when they are due.*

Assumptions

1. Library has or is willing to purchase an AWS Account
2. Library has a pre-existing domain name
3. The libraries inventory of items exists in CSV or some other format that can easily be loaded into a database table
4. The libraries list of customers and employees (aka “Users”) exists in CSV or some other format that can easily be loaded into a database table
5. Library can afford to hire a graphic artist to handle styling and layout of the frontend pages
6. There is no limit on the number of articles (or items) that a single user can checkout at one time
7. Late fee charges are not recorded or stored by the system
8. The length of time between the loan date and due date of a library article is not defined by the system
9. At the outset, there are 3 types of users: “Customers”, “Operators”, “Managers”
10. Users are created by Operators
11. Developer is familiar with Assignment 3 ☺ (I can only write so much here)

Entity Model



Based on the above diagram, the following database objects will be required for each model element:

- Database Table
- Insert Procedure
- Update Procedure
- Delete Procedure

The schema for the tables and requisite procedures is as follows.

ArticleType Schema

This is a lookup table that is used to describe the type of the library article or item. At the outset it include 3 records: “Book”, “Disk”, “Magazine”.

Field	Type	Constraint	Key	Foreign Key	Default
ArticleType_ID	BigInt	Unique, Not Null	Y		
Name	Varchar(255)	Unique, Not Null	N		

ArticleSection Schema

This is a lookup table that used to describe the section of the library that a given article belongs too.

Field	Type	Constraint	Key	Foreign Key	Default
ArticleSection_ID	BigInt	Not Null, Unique	Y		
Name	Varchar(255)	Not Null, Unique	N		

Article Schema

This is one of the main tables of the database. Each record corresponds to a specific item in the library's collection.

Field	Type	Constraint	Key	Foreign Key	Default
Article_ID	BigInt	Unique	Y		
ArticleType_ID	BigInt	Not Null	N	ArticleType	
ArticleSection_ID	BigInt	Not Null	N	ArticleSection	
IsRetired	Bit	Not Null	N		FALSE
Title	Varchar(255)	Not Null	N		
Author	Varchar(255)	Not Null	N		Unknown
DeweyDec	Varchar(255)	Not Null	N		000

User Schema

This table defines all uses of the inventory system. This includes both employees and customers.

Field	Type	Constraint	Key	Foreign Key	Default
User_ID	BigInt	Not Null, Unique	Y		
FirstName	Varchar(255)	Not Null	N		
LastName	Varchar(255)	Not Null	N		
SocialSecurityID	Varchar(255)	Not Null, Unique	N		
Active	Bit	Not Null	N		TRUE
AddressLine1	Varchar(255)	Not Null	N		
AddressLine2	Varchar(255)		N		
PhoneNumber	Varchar(255)	Not Null	N		
City	Varchar(255)	Not Null	N		
ZipCode	Varch(10)	Not Null	N		
BirthDate	Date	Not Null	N		
FirstName	Varchar(255)	Not Null	N		
LastName	Varchar(255)	Not Null	N		
Password	Varchar(40)	Not Null	N		
Locked	Bit	Not Null	N		FALSE
UpdatePassword	Bit	Not Null	N		TRUE

Note – The "Locked" bit is used to lock a user out of the system completely. The UpdatePassword flag is used to force a user to

enter a new password. This flag is set to true when a new user is added to the system.

UserArticleLink Schema

This table is used to link users to articles. One user can map to "n" articles for a given "Loan Date" / "Due Date" combination.

Field	Type	Constraint	Key	Foreign Key	Default
UserArticleLink_ID	BigInt	Unique, Not Null	Y		
Article_ID	BigInt	Not Null	Y	Article	
User_ID	BigInt	Not Null	N		
LoanDate	Date	Not Null	Y		Now
DueDate	Date	Not Null	Y		

Role Schema

This table is used to describe roles (or functions) assigned to groups of users. This table drives what pieces of the frontend will be visible to users upon login. Each of the *Admin bit flags will be used to determine whether a given user can access the CRUD (Admin Interface) for the associated Model entity. For instance, if a user (or group of users) map to a role with the "isArticleSectionAdmin" flag set to true, then they will be able to access the ArticleSection Admin interface.

Field	Type	Constraint	Key	Foreign Key	Default
Role_ID	Int	Unique, Not Null	Y		
Name	Varchar(255)	Unique, Not Null	N		
isArticleAdmin	Bit	Not Null	N		FALSE
isArticleTypeAdmin	Bit	Not Null	N		FALSE
isArticleSectionAdmin	Bit	Not Null	N		FALSE
isRoleAdmin	Bit	Not Null	N		FALSE
isUserAdmin	Bit	Not Null	N		FALSE
SeeReports	Bit	Not Null	N		FALSE

The "SeeReports" flag is used to drive whether a given user will be able to access certain specialty reports.

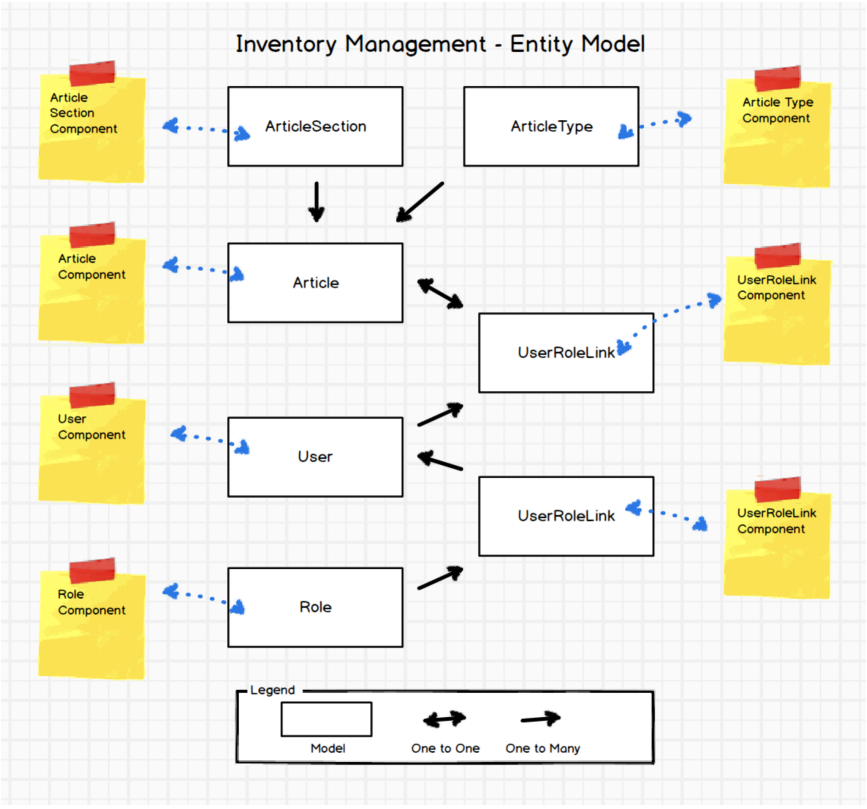
UserRoleLink

This table is used to link users to roles. One user can map to "n" roles.

Field	Type	Constraint	Key	Foreign Key	Default
UserRoleLink_ID	Int	Unique, Not Null	Y		
User_ID	BigInt	Not Null	N	User	
Role_ID	Int	Not Null	Y	Role	

MVC Elements and their relationships

The frontend pieces of the application should follow the organization of the entity model. Specifically, it should be broken down into components based on the entity model.



The Article sub-domain should be the default (or entry point) for the application. It should provide–

- non-authenticated users with the ability to search all articles available in the library
- Authentication
- Access to the other sub-domains based on the user's role
- Define a "true-access" variable, which is then used across all other components to prevent fishing
- Force users with isNew flag to true to update their password

All sub-domains follow a similar MVC organization and structure. Specifically–

1) (Root) Master Controller Script (controller.php) – Entry point to all component logic. It handles–
   a. The HTTP GET/POST Requests and routes the request to the appropriate view.
   b. Loading all model scripts into Memory
   c. Verifies that database is up running

2) Model Sub-Directory – One Script is defined for each model accessed by the component. This script is then used to wrap specific stored procedures needed by the component. There should be no hard coded SQL written into the scripts. This directory should also include a singleton database connection function, which is used across all wrapper methods to open a database connection.

3) View Sub-Directory – In most cases only a single script will be needed. This script should define a single entry point function called "view", which is invoked by the Master Controller (1).

4) All php scripts should begin by evaluating whether the "true-access" variable is defined. Specifically, line one should be as follows:

```
<?php defined("true-access") or die;
  …
```

5) config.php (Root) – This script should be used to define global constant and database connection properties. It should also define the salt value applied when a user's password is reset.

Major Visual Layout Considerations

The layout of the site should be Mobile device compatible. To this end, it is recommended that a secondary framework such as ExtJS Touch, or JQUERY Mobile be used. Whatever framework is chosen, it should be used consistently across all sub-domains (components) of the site. In addition, all PHP that is used to write javaScript or html should be defined using a single "base.php" script. This script should be defined in the root directory of the application and re-used across components.

Security and Access Control

As already mentioned, Users are defined in the User database table. Their access to specific components is driven by the roles that their account maps to in the UserRoleLink table.

Deployment Strategy

The application will be deployed using bash shell scripts that leverage the AWS EC2 API. To this end, the developer will need to deliver following files:

1) Master Database Install script - The developer should define all database objects using a master SQL script. This script should initialize the database and stage all necessary baseline data.
2) Zipped Web Server Directory – A zip containing all component files that should be deployed to the tomcat apache server