

Student id: 100001252

Student name: Mansour shadloo

Swinburne universtity of Technology

Subject code: COS80011

Subject Title: Web application architectures

Assignment number: assingment 01

Due Date: 23 April 2017, 12:00pm

Lab Group: Tuesday 19:30-21:30

Lecturer: Prof Jun Han

Contents

[Design and operation of each component 4](#_Toc481169720)

[Book Inventory Service(Web service) 4](#_Toc481169721)

[Student Soap Service(not a web service) 8](#_Toc481169722)

[Booking application 9](#_Toc481169723)

[Structure of Database 10](#_Toc481169724)

[ERD Diagram 10](#_Toc481169725)

[Server01 10](#_Toc481169726)

[Server02 10](#_Toc481169727)

[Tables Schema 11](#_Toc481169728)

[How tables update 12](#_Toc481169729)

[UML Collaboration diagram 13](#_Toc481169730)

[Static model 13](#_Toc481169731)

[View All Books 14](#_Toc481169732)

[Network diagram 14](#_Toc481169733)

[View student Records 14](#_Toc481169734)

[Network diagram 15](#_Toc481169735)

[Borrow a book (all studentand books already loaded in arraylists in university object) 15](#_Toc481169736)

[Request a book 15](#_Toc481169737)

# Design and operation of each component

## Book Inventory Service(Web service)

Book inventory service is a resfull api that connect with database and get some information from database. Book inventory service has access to two tables:

1. Books Table

2. Student borrow Table

This service use XML to transport message .Main class that runs all services for book inventory called UniLibBook. Xml file help service to create url for web api. Inventory Service has some functions as below to run services:

/\*\* get all book in unilib

\*

\* **@return** all book in String format

\*/

**public** String getAllBook()

**endpoint**: http://192.168.1.229:9763/services/UniLibBook.UniLibBookHttpEndpoint/

**weburl**: <http://192.168.1.229:9763/services/UniLibBook.UniLibBookHttpEndpoint/Books>

/\*\*get records of one specific student

\*

\* **@param** studentId

\* **@return** student details in String format

\*/

**public** String getRecordsStudent(Integer studentId)

**endpoint**: http://192.168.1.229:9763/services/UniLibBook.UniLibBookHttpEndpoint/

**weburl**: <http://192.168.1.229:9763/services/UniLibBook.UniLibBookHttpEndpoint/student-borrow/%3F>

/\*\*get records of one specific student just different version and it gives more infor

\*

\* **@param** studentId

\* **@return** student details in String format

\*/

**public** String getRecordsStudent2(Integer studentId)

/\*\*get all student details

\*

\* **@param** studentId

\* **@return** all student details in string format

\*/

**public** String getAllStudentRecords()

**endpoint**: http://192.168.1.229:9763/services/UniLibBook.UniLibBookHttpEndpoint/

/\*\*view details of a specific book by book id

\*

\* **@param** bookId

\* **@return** detail of that specific book in string format

\*/

**public** String getBook(Integer bookId)

/\*\*update book detail by bookid

\*

\* **@param** bookid

\* **@param** title

\* **@param** authorList

\* **@param** isbn

\* **@param** publishDate

\* **@param** status

\* **@return** String message if it is successfull

\* **@throws** BookException

\*/

**public** String updateBook(Integer bookid,String title,String authorList,String isbn,String publishDate,String publisher,String status) **throws** BookException

/\*\*function to add book with all details

\*

\* **@param** bookTitle

\* **@param** authorList

\* **@param** isbn

\* **@param** publisher

\* **@param** publishDate

\* **@param** status

\* **@return** string of successful or error message

\* **@throws** BookException

\*/

**public** String addBook(String bookTitle,String authorList,String isbn,String publisher,String

publishDate,String status) **throws** BookException

/\*\*easier way of adding book just for test other elements are default

\*

\* **@param** bookTitle

\* **@param** isbn

\* **@return** String that shows if book inserted successfully

\* **@throws** BookException

\*/

**public** String addBook2(String bookTitle,String isbn) **throws** BookException

/\*\*delete the book with book id

\*

\* **@param** bookId

\* **@return** if it is deleted shows bookid... deleted successfully

\* **@throws** BookException

\*/

**public** String deleteBook(Integer bookId) **throws** BookException

/\*\* action in database has to be borrow request or return (although if a student return the book we delete the that record)

\*

\* **@param** action

\* **@return**

\*/

**private** **boolean** isActionValid(String action)

/\*\*update current student record search by studentid and bookid and change action(e.g to returned)

\* it just change the action if it exist

\* **@param** studentId

\* **@param** bookId

\* **@param** action

\* **@return** if all good it shows action successfully in string format

\* **@throws** BookException

\*/

**public** String updateStudentrecords(Integer studentId,Integer bookId,String action) **throws** BookException

/\*\* function to insert student borrow record. so they can update if they have existing record

\*

\* **@param** studentId

\* **@param** bookId

\* **@param** action

\* **@return** if it insert successfully it return successfull message

\* **@throws** BookException

\*/

**public** String insertStudentrecords(Integer studentId,Integer bookId,String action) **throws** BookException

/\*\* delete student record if it exist for that specific studentId and bookId

\*

\* **@param** studentId

\* **@param** bookId

\* **@return** successfull message if delete student records successfully

\*/

**public** String deleteStudentrecords(Integer studentId,Integer bookId)

/\*\*function to setBookStatus in case that you gonna change status

\*

\* **@param** bookId

\* **@param** bstatus

\* **@return** return successfull if it set status for bookdb

\* **@throws** BookException

\*/

**public** String setBookStatus(String bookId,String bstatus) **throws** BookException

<service name=*"UniLibBook"* scope=*"application"*>

<description>

Please Type your service description here

</description>

<transports>

<transport>http</transport>

</transports>

<messageReceivers>

<messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-only"* class=*"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"* />

<messageReceiver mep=*"http://www.w3.org/2004/08/wsdl/in-out"* class=*"org.apache.axis2.rpc.receivers.RPCMessageReceiver"*/>

</messageReceivers>

<parameter locked=*"false"* name=*"ServiceClass"*>au.edu.swin.waa.UniLibBook</parameter>

<parameter name=*"disableSOAP11"*>true</parameter>

<parameter name=*"disableSOAP12"*>true</parameter>

<operation name=*"addBook"*>

<parameter name=*"RESTMethod"*>POST</parameter>

<parameter name=*"RESTLocation"*>Books</parameter>

</operation>

<operation name=*"addBook2"*>

<parameter name=*"RESTMethod"*>POST</parameter>

<parameter name=*"RESTLocation"*>Books2</parameter>

</operation>

<operation name=*"updateBook"*>

<parameter name=*"RESTMethod"*>PUT</parameter>

<parameter name=*"RESTLocation"*>Book/{bookId}</parameter>

</operation>

<operation name=*"updateStudentrecords"*>

<parameter name=*"RESTMethod"*>PUT</parameter>

<parameter name=*"RESTLocation"*>studentRecord/{studentId}/{bookId}</parameter>

</operation>

<operation name=*"insertStudentrecords"*>

<parameter name=*"RESTMethod"*>POST</parameter>

<parameter name=*"RESTLocation"*>studentRecords</parameter>

</operation>

<operation name=*"deleteStudentrecords"*>

<parameter name=*"RESTMethod"*>POST</parameter>

<parameter name=*"RESTLocation"*>studentRecord/{studentId}/{bookId}</parameter>

</operation>

<operation name=*"setBookStatus"*>

<parameter name=*"RESTMethod"*>PUT</parameter>

<parameter name=*"RESTLocation"*>book/{bookId}</parameter>

</operation>

<operation name=*"deleteBook"*>

<parameter name=*"RESTMethod"*>DELETE</parameter>

<parameter name=*"RESTLocation"*>Book\{bookId}</parameter>

</operation>

<operation name=*"getBook"*>

<parameter name=*"RESTMethod"*>GET</parameter>

<parameter name=*"RESTLocation"*>Book/{bookId}</parameter>

</operation>

<operation name=*"getRecordsStudent"*>

<parameter name=*"RESTMethod"*>GET</parameter>

<parameter name=*"RESTLocation"*>student-borrow/{studentId}</parameter>

</operation>

<operation name=*"getRecordsStudent2"*>

<parameter name=*"RESTMethod"*>GET</parameter>

<parameter name=*"RESTLocation"*>student-borrow2/{studentId}</parameter>

</operation>

<operation name=*"getAllBook"*>

<parameter name=*"RESTMethod"*>GET</parameter>

<parameter name=*"RESTLocation"*>Books</parameter>

</operation>

<operation name=*"deleteBook"*>

<parameter name=*"RESTMethod"*>DELETE</parameter>

<parameter name=*"RESTLocation"*>book/{bookId}</parameter>

</operation>

</service>

## Student Soap Service(not a web service)

Student soap service is a Soap api that connect with database and get some information from database. Student soap service has access to one table/database:

1. Students Table

Assumption is that this server is separate from other service(Restfull services), so we cannot use any inner join with book table to get information. This service use XML to transport message .Main class that runs all services for book inventory called UniLibStudent. Student soap Service has some functions as below to run services:

endpoint: http://192.168.1.229:9763/services/UniLibStudent.UniLibStudentHttpSoap11Endpoint/

/\*\*this function get all student and return in format of string

\*

\* **@return** string all student details line by line(it has been used inside application)

\*/

**public** String getAllStudent()

/\*\*this function add a new student into database

\*

\* **@param** fullName

\* **@param** pin

\* **@return**

\* **@throws** StudentException

\*/

**public** String addStudent(String fullName,String pin) **throws** StudentException

/\*\* this function remove student from database

\*

\* **@param** studentId

\* **@return**

\* **@throws** StudentException

\*/

**public** String deleteStudent(Integer studentId) **throws** StudentException

/\*\*this function get student via studentId

\*

\* **@param** studentId

\* **@return**

\*/

**public** String getStudent(Integer studentId)

/\*\*this function is for authentication it checks

\* if student with the student id and pin exists in database

\*

\* **@param** studentId

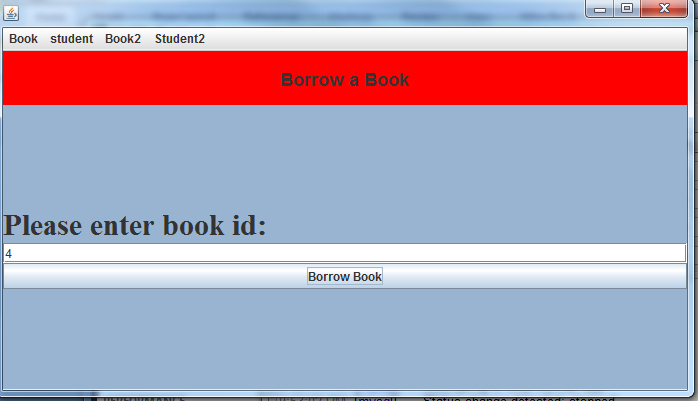
\* **@param** pin

\* **@return**

\*/

**public** String isStudentExist(Integer studentId,String pin)

## Booking application



For booking application GUI has been used. For all functionality that needed to create for this assignment there are four main menu on the menubar. Under book menu there are two main functions :

1. View all books: shows all books with their detail and status, and it is using resfull inventory services.

2. View Student records: show records of borrowing and buying for a specific student. It starts with checking student authentication.

Under Student menu there are two main functions:

1. Borrow a book: start with student authentication, and then by the end let user to borrow a book.

2. Request a book: start with Student authentication, and then by the end let user to request a new book if it does not exist in library.

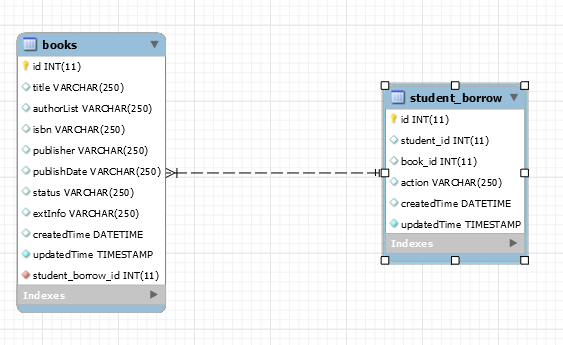
# Structure of Database

One of the aspects of any software or web/api application is database structure. In this project as we have small block of works we have three main tables, and we assume that book, student borrow records are in one database in lets say server1 and student table in server02 and Google database is connected with Google api and it is not under our control.

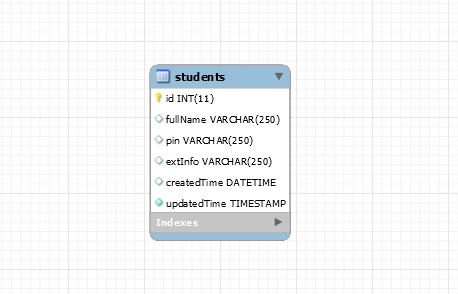
## ERD Diagram

As we can see below in diagram, there is no relationship between borrowing records and student(as it is in different server), and inner join is not allowed between these tables.

### Server01



### Server02



## Tables Schema

`student\_borrow` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`student\_id` INT(11) NULL DEFAULT NULL,

`book\_id` INT(11) NULL DEFAULT NULL foreign key references waa.books,

`action` VARCHAR(250) NULL DEFAULT NULL,

`createdTime` DATETIME NULL DEFAULT NULL,

`updatedTime` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`),

UNIQUE INDEX `student\_borrow\_uniquekey` (`book\_id` ASC))

-- -----------------------------------------------------

-- Table `waa`.`books`

-- -----------------------------------------------------

`books` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`title` VARCHAR(250) NULL DEFAULT NULL,

`authorList` VARCHAR(250) NULL DEFAULT NULL,

`isbn` VARCHAR(250) NULL DEFAULT NULL,

`publisher` VARCHAR(250) NULL DEFAULT NULL,

`publishDate` VARCHAR(250) NULL DEFAULT NULL,

`status` VARCHAR(250) NULL DEFAULT NULL,

`extInfo` VARCHAR(250) NULL DEFAULT NULL,

`createdTime` DATETIME NULL DEFAULT NULL,

`updatedTime` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

`student\_borrow\_id` INT(11) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `uq\_book\_cons` (`isbn` ASC),

-- -----------------------------------------------------

`waa`.`students`

-- -----------------------------------------------------

`students` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`fullName` VARCHAR(250) NULL DEFAULT NULL,

`pin` VARCHAR(250) NULL DEFAULT NULL,

`extInfo` VARCHAR(250) NULL DEFAULT NULL,

`createdTime` DATETIME NULL DEFAULT NULL,

`updatedTime` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`))

## How tables update

The way that tables update follows as below:

1. **Student borrow a book:** Status of book in books table will be changed to on loan, and in student records student\_id and book\_id insert as new row with action=on loan

2. **Student request a book:** A new book will be inserted into books table with status back order. Then in borrowing record one new row will be inserted with student\_id and book\_id and action= back order

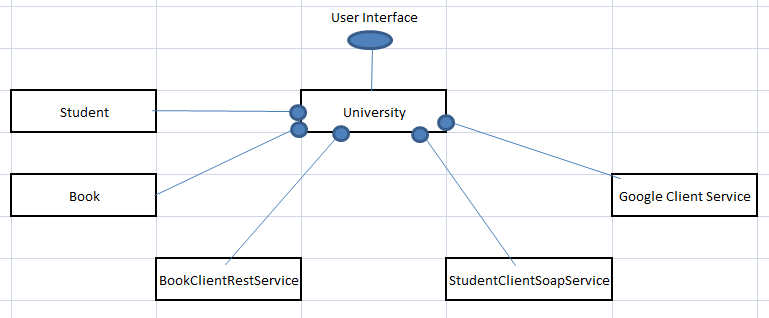
3. **Student return a borrowed book:** Status of the book will be changed to available in books table and row with book\_id will be deleted from student\_borrow table

4. **Requested book become available:** status of book in books table will be changed to available. Record with book\_id will be deleted from student borrowing table, and student will be notified that book is now available in library for loan.

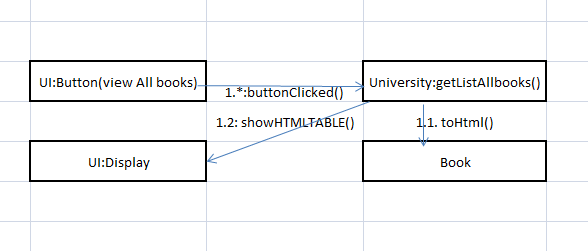
# UML Collaboration diagram

Initially when application start working, it loads all data from database into objects, and objects list. Below is UML diagram that shows how I loaded all database into objects, when application start working:

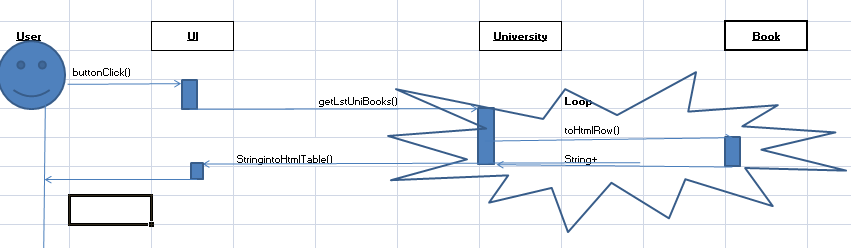
## Static model



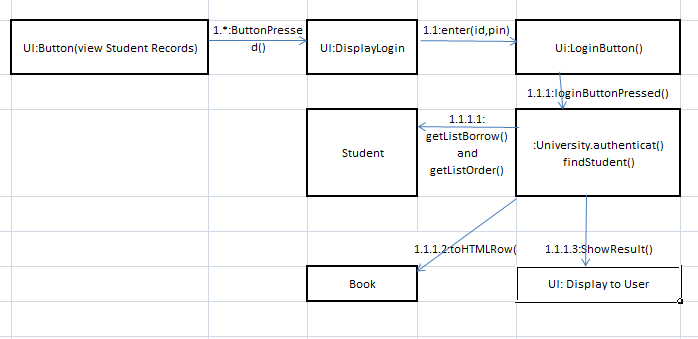
## View All Books



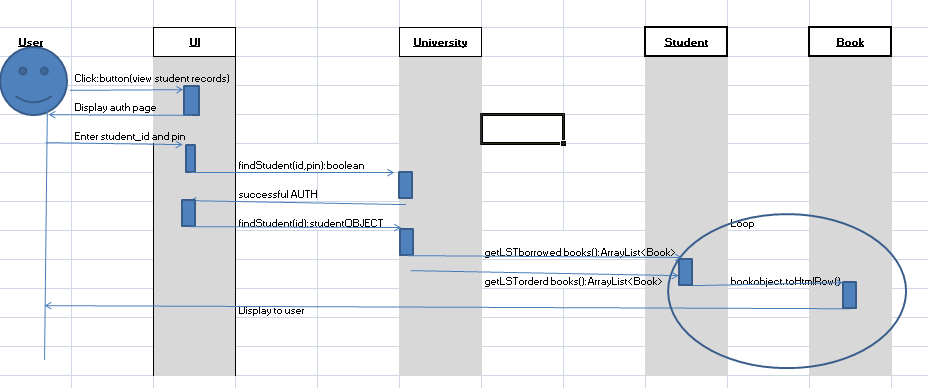
### Network diagram



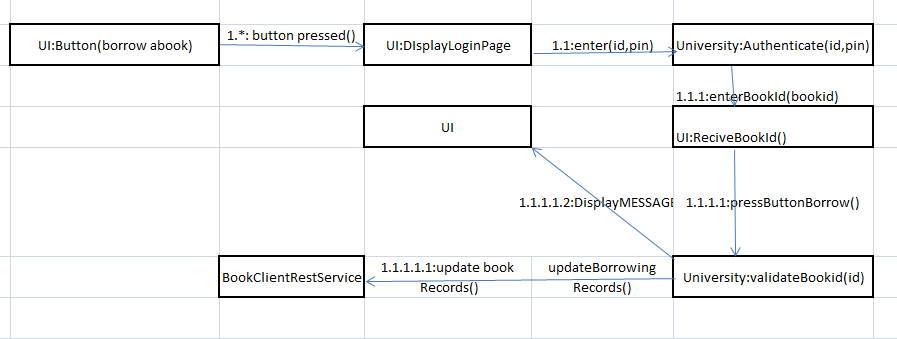
## View student Records



### Network diagram



## Borrow a book (all studentand books already loaded in arraylists in university object)



## Request a book

