Sheffield Cloudbase Platform

Team 05:

Sanika Vikas Patil

Vijeta Agrawal

Seham Alharbi

Chathura Dilshan Gulavita Ganithage

Public URL:

http://143.167.9.201:8080/SingleSignIn/

Content Index:

Figure Index:

Table Index:

1. Introduction

Platform as a Service (PaaS) is a cloud computing service that allows the user to develop, run and manage applications. It eliminates the complexity of building and maintaining the infrastructure associated with developing and launching an app. Here we are developing a PaaS named **Sheffield Cloudbase**, which is based on the theme named **Academia,** useful for the students of University of Sheffield. It will allow them to use the applications hosted on it and at the same time they can deploy any of their own applications which meets a required criterion.

Sheffield Cloudbase consists of two microservices (Single Sign on and Peanut bank account) and two pre-hosted applications (Library and ASK application). Moreover, it will allow any Independent Software Vendors (ISVs) to deploy their applications which are relevant to the theme of our platform.

2. Business

The theme of the PaaS that our team is developing is Academia, and therefore it will only contain student-themed applications that will facilitate academic support for the university students. Initially the team members proposed to comply with “life-style theme” but considering students requirements, the team came to a common ground to provide academic support by developing and deploying appropriate web-based applications. In addition, the consistent business theme of Sheffield Cloudbase will only allow the hosting of similar applications developed by the members of other teams who will act as independent software vendors (ISVs). The default applications which our platform will be rendering are as follows:

1. The first application is named **Library**. It provides a user-friendly way for students to search for a book in an efficient way and check status of the books that they are looking for. It also provides the user to reserve a particular book for a limited amount of time(3 days).Additionally, it facilitates the users to view the popularity of the books amongst other readers by viewing the reviews and creating their own.

2. The second application is called **ASK**. In this application, the university students will be able to post questions related to multiple academic skills from different departments. These questions, thus, can be subsequently answered by other students. And the students will get the chance to share and learn the skills which are not only related to their department but also other departments.

During the process of developing a product from scratch, we have to brainstorm to finalize the technology stack to provide the best user experience. This is the significant step while developing a product. Developer has to consider many factors as Speed and Performance, Security, Market life cycle for that technology, long term vendor support etc. while arriving at a particular decision.

The technologies which we considered for programming were Python and Java. Python frameworks are gaining popularity for developing dynamic web applications. It is not only easy to learn but also decreases the time for development with its easy-to-read syntax and simple compilation feature. Similarly, we also had discussion on using Node.js, which runs javascript applications on both server and client side. These technologies also support various template engines, which enables the developer to use static template files in their applications.

But since the server side uses Apache Tomcat Server which implements Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies: we decided to use Java as our programming language.

While considering data storage, we had MySQL and MongoDB in our mind. When we read about these databases, we found that from authentication point of view, both the databases are secured. Also, MongoDB follows a very straightforward and common authorization model:role-based access control. So, for Login purpose this database would suffice the need of not allowing the system to be accessed for the user outside the role assignment. But as our server has MySQL installed in it, we would continue to use the same.

Inorder to make our microservices to communicate with each other, we thought of REST (**RE**presentational **S**tate **T**ransfer) APIs and service discovery architectural styles. To invoke a service using REST APIs, our code should know the network location (IP address

and port) of the service instance. But in cloud-based microservices, service instances are dynamically assigned network locations. Moreover, the set of service instances changes dynamically because of autoscaling, failures, and upgrades. Consequently, our client code needs to use a more elaborate service discovery mechanism. In such case we can use service discovery architectural style.

There was disagreement on which architectural style should be used. Then, after detail examination on these two architectural styles, we came to the conclusion that our platform will be running on the physical hardware, where the network locations of service instances will be relatively static. So, using RESTful APIs for establishing communication between our microservices would suffice the need.

The team has agreed on dividing the work between its members in the following way:

1. **Sanika Patil:** Single Sign In microservice.
2. **Vijeta Agrawal:** Payment microservice.
3. **Chathura Dilshan Gulavita Ganithage:** Library application.
4. **Seham Alharbi:** Ask application.

3. Design

3.1 Technologies and its risk analysis

The **web technologies** that were taken into consideration were HTML, W3.CSS and JSP (Java Server Pages) to create dynamically generated web pages based on HTML.

But we also thought of using template engines as it had many advantages over JSP. The template engines are useful in creating dynamic web pages by allowing developers to generate desired content types, such as HTML, while using some of the data and programming constructs such as conditionals and for loops to manipulate the output. The alternatives we considered for our presentation layer were template engines like “freemaker” and “velocity”. But as we have limited memory on our VM, we decide to go with JSP.

When it comes to develop the applications using different Java technologies like Spring, Hibernate etc., team members are free to proceed the way that suits them. We have not agreed on a specific java technology to be used. But as a part of our research, we found that Spring make use of the idea of "Inversion of Control and Dependency injection" in an efficient, easy and best possible ways while developing application. It also provides secure way to handle Login. Spring handles “Autowiring”, which becomes very difficult while dealing with complex applications. Also, all the dependencies are taken care of, which the developer needs for building the application (Spring.io., 2019).

Also, MVC (Model-View-Controller) architectural pattern encourages good separation between the actual view logic and rendered view. The separation of view from the logic makes it easy to update the look and feel of the application, rather than having to re-write the source code (Docs.spring.io., 2019).

Spring MVC is one of many frameworks built on top of **servlets** to try make the task of writing a web application a bit easier. Having all these advantages, some team members are willing to use these approaches to develop an application. It will depend on the preference of individual team member and also taking VM memory into consideration (Spring.io., 2019).

Tomcat server is an open-source java servlet container which is run on a Java Virtual Machine. As we are using Tomcat server, we should take into consideration that the changes from the development to the deployment environment might throw errors if the dependencies between them are not satisfied. Furthermore, the file permissions might need to be changed while deploying our PaaS as we will have root permissions during development which might not be the case while deployment.

3.2 Platform Architecture

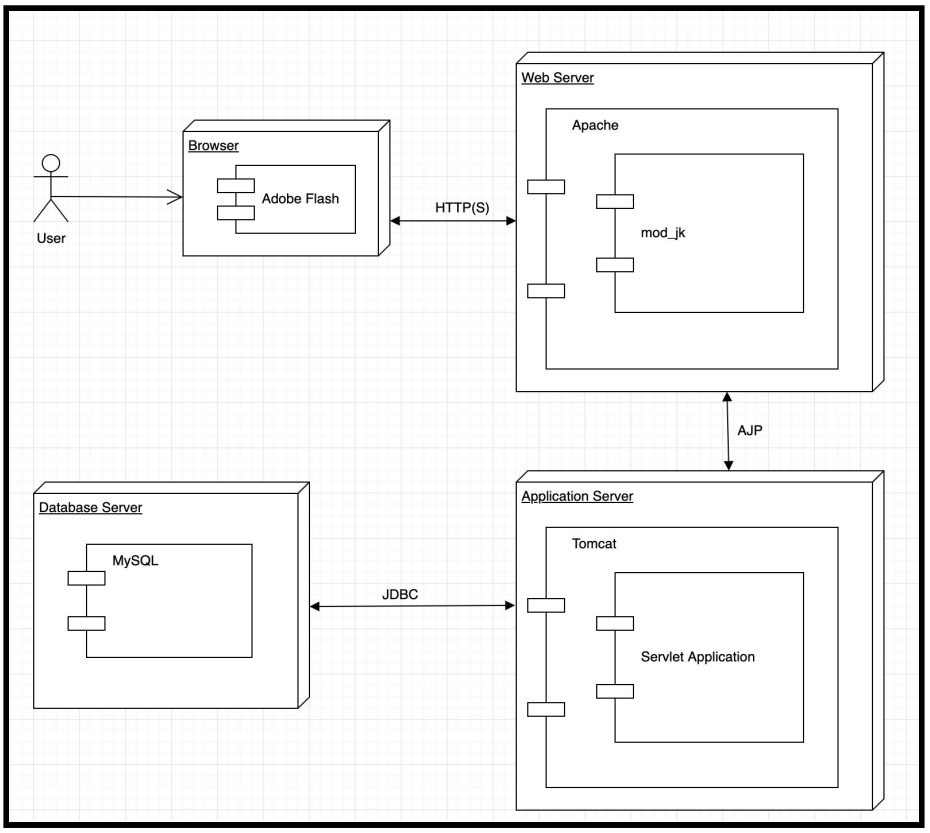


Figure 1: PaaS Architecture

Above figure displays the UML deployment diagram of our PaaS. It shows the connections established when we deploy Sheffield Cloudbase: Servlet Application on Apache Tomcat Server. The user will access Sheffield Cloudbase through client side: Browser which communicates with the web server using HTTP(S) requests. Web server communicates with application server using AJP and application server communicates with database server using JDBC. Some Components are explained below for better understanding:

Mod_jk: It is an Apache module used to connect the Tomcat servlet container with web servers.

JDBC: It provides an abstraction layer between Java applications and database servers, so that an application's code does not need to be altered in order for it to communicate with multiple database formats. Rather than connecting to the database directly, the applications send requests to the JDBC API, which in turn communicates with the specified database through a driver that converts the API calls into the proper dialect for the database to understand. If a developer wishes to access two different database formats in the same program, they don't need to add any additional syntax to their code; they simply call two different JDBC drivers (MuleSoft. (2019)).

AJP: The Apache JServ Protocol (AJP) is a binary protocol that can proxy inbound requests from a web server through to an application server that sits behind the web server. Web implementors typically use AJP in a load-balanced deployment where one or more front-end web servers feed requests into one or more application servers. Sessions are redirected to the correct application server using a routing mechanism wherein each application server instance gets a name (called a route). In this scenario the web server functions as a reverse proxy for the application server. Lastly, AJP supports request attributes which, when populated with environment-specific settings in the reverse proxy, provides for secure communication between the reverse proxy and application server (En.wikipedia.org. (2019)).

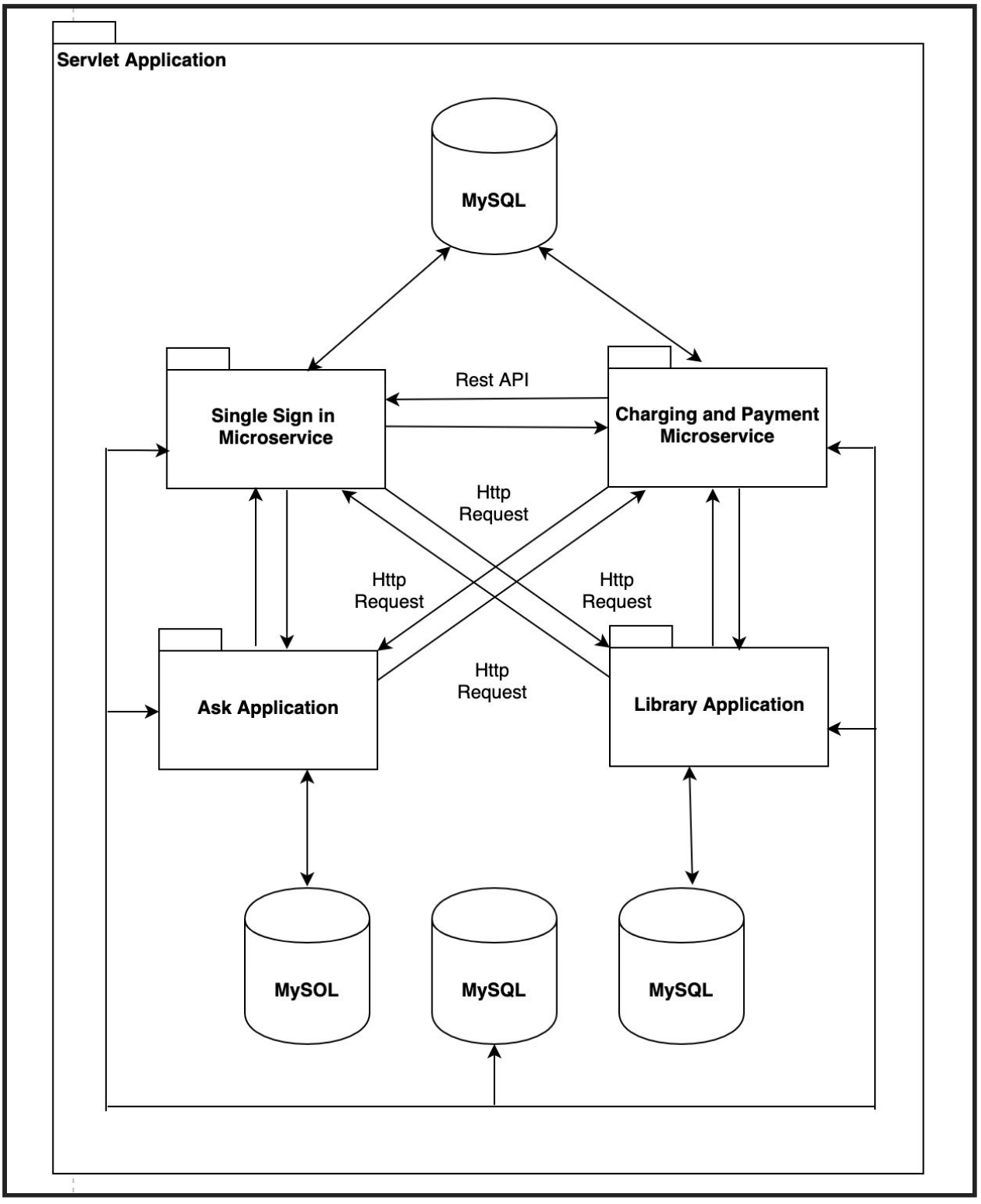


Figure 2:UML Package Diagram

Figure 2 displays the UML package diagram of Sheffield Cloudbase. It shows how our microservices and applications are connected to each other. The microservices

are using same database, while the applications are using separate databases. The additional database is used to store the information of the logged in user like sessionId, userId, username and role. This database is shared among all the microservices and applications. The microservices are communicating with each other using Restful service. The applications are communicating with the microservices using Http Request.

Note: The architecture has been changed from the previous mentioned architecture, as the application development members have not implemented using Restful service as discussed during design phase. Therefore, the microservice are developed using Restful service and the applications are developed using Http Requests.

3.3 App directory structure

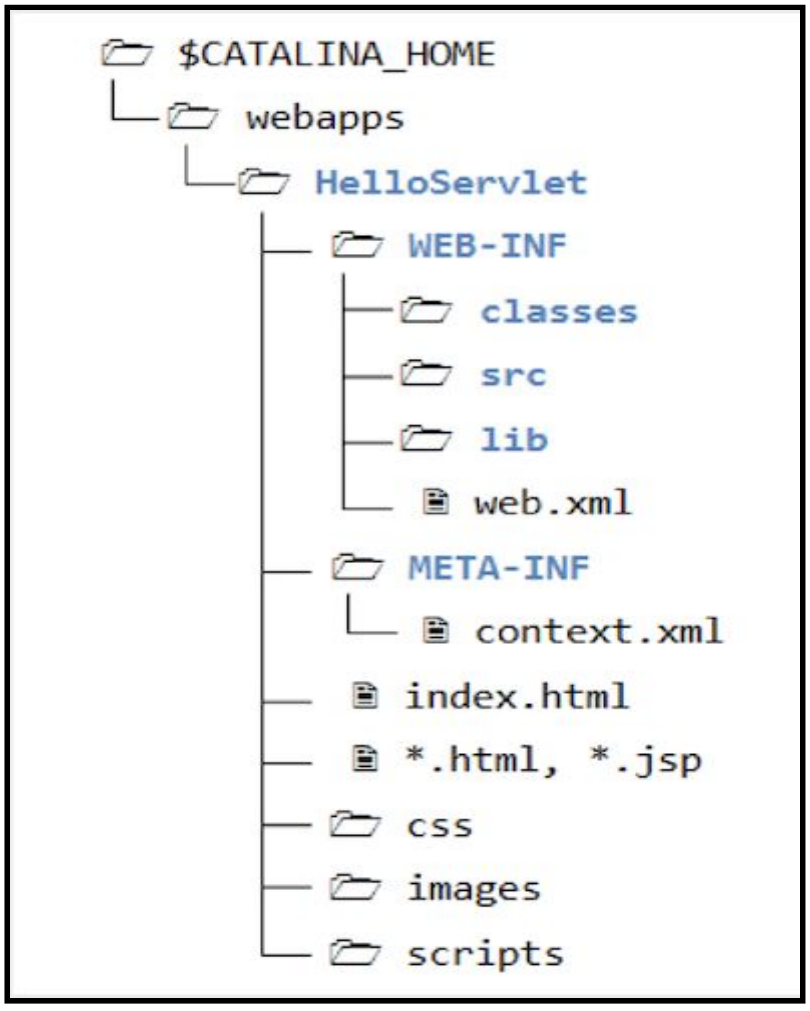


Figure 3:App Directory Structure

The structure of the (.war) directory that will be used to store an application is divided into other sub-directories and files as follows:

1. The App's context path: which is the directory that will have the same name as the application name.
2. WEB-INF: this directory will be placed inside the App's context path, and it will contain all .class and .java files.
3. The application home page (index.html) and other JSP and HTML pages.

This WAR-file can be uploaded to the platform by simply copying it and placing it under the (webapps) directory that is located under $CATALINA_BASE. Moreover, this location will give the Tomcat server the ability to unpack the (.war) file automatically.

4. Technical

4.1 Single Sign In microservice

The Single Sign In microservice is implemented using Spring MVC framework. Figure 4. shows the architecture used to develop the microservice. MVC (Model-View-Controller) architectural pattern foster good separation between the actual view logic and rendered view. JdbcTemplate is used for JDBC data access, which is a abstraction layer to have communication between the single sign in microservice and MySQL database server.

Single sign in microservice uses three user roles:

1. Admin
2. ISV
3. User

According to the roles of the logged in users, they are authenticated and authorized to use the corresponding dashboards as shown in figures below. When the

user is successfully logged in, it is added in the http session and the session is terminated once the user is logged out.

Security is an important concern which should always be taken care of. To address the security issue, the passwords are not directly stored in plain text in database. They are encrypted before storing in the database using cryptographic algorithm: DES. The secret key is hashed using message digest algorithm:SHA. Thus even if the hacker gets the key, he would not be able to decrypt the passwords. The length of the password that the user can register with, is restricted to be exactly 8 letters. This restriction has been put, as it gives encryption error if the password length is not multiple of 8. Thus for example, having 16 letters would have been too long, so I restricted it to 8 letters.

For authentication, the hash function is applied on the password which user will enter and then compared with the hash stored in the database. If both hashes match then the user is authenticated, since hash of same input will give same output.

Connection pooling is taken care by using DataSource objects. All of the connections produced by instances of that DataSource class will automatically be pooled connections. This will provide connection from the pool of database connections, instead of creating new connection during every web requests. Thus it will help to improve performance. In this microservice, connection pooling is done by using Java Database Connectivity (JDBC) API and the Apache DBCP pooling library.

Dashboard is connected to the single sign in microservice. The upload feature for ISV implements two functionalities,

1. Uploading .war file: This will upload the application to the desired location on server.
2. Uploading .sql file (SQL script): This will upload the sql script on some temporary location, run and delete the script. The script is deleted to free the memory. In order to run the SQL script , iBATIS framework is used. It automates the mapping between SQL databases and java objects. The functionality is implemented using “ScriptRunner” class.

The code has been checked by team member - Vijeta Agrawal, to make sure that is meets the code artefacts and also is readable to other developers.

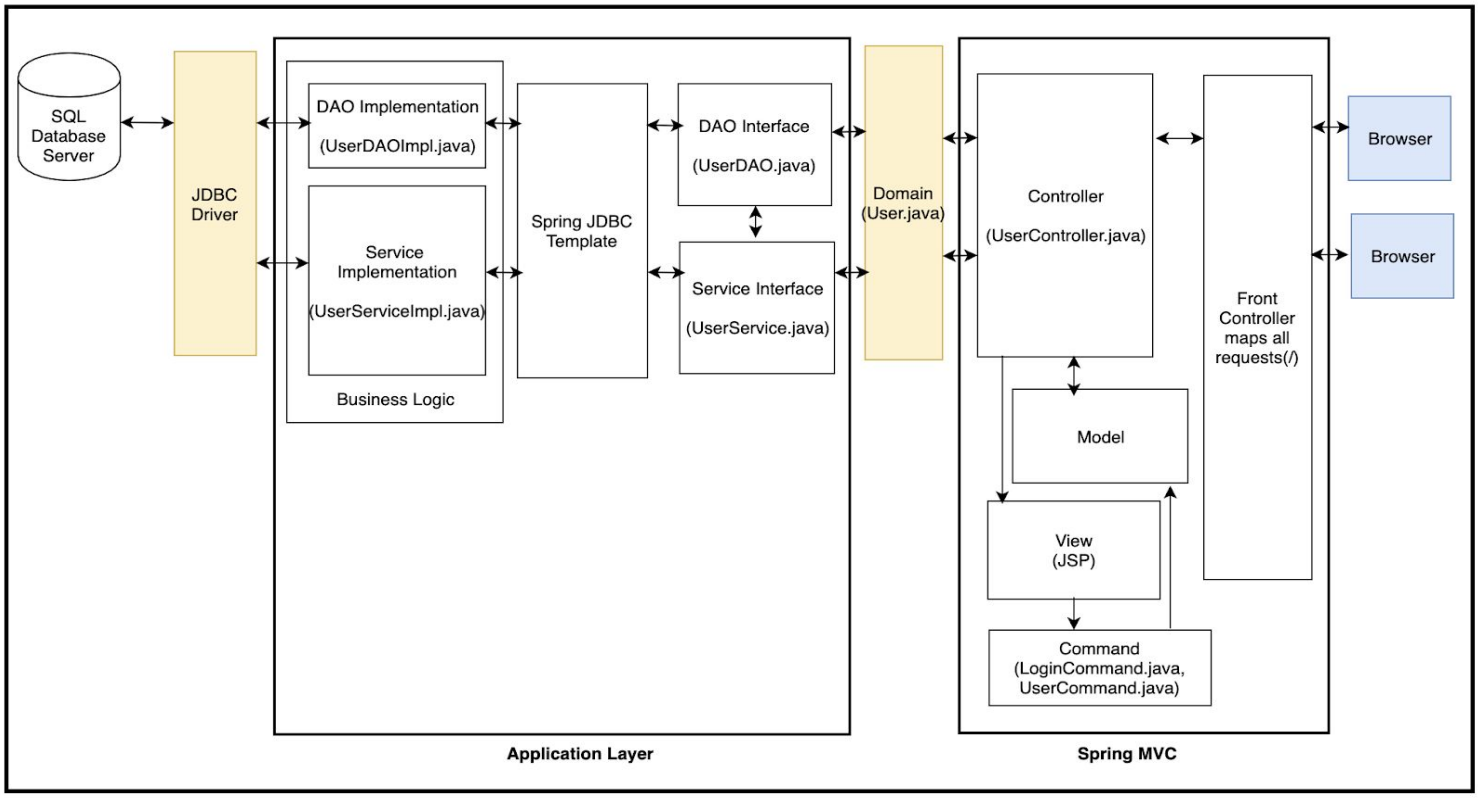


Figure 4:Single Sign In microservice architecture

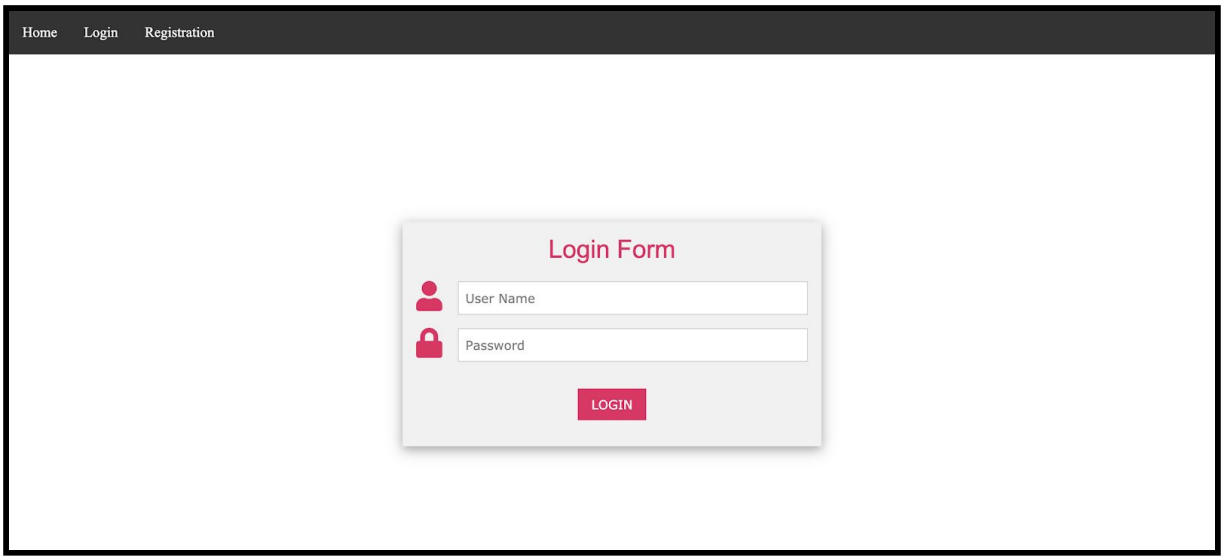


Figure 5:Login Form

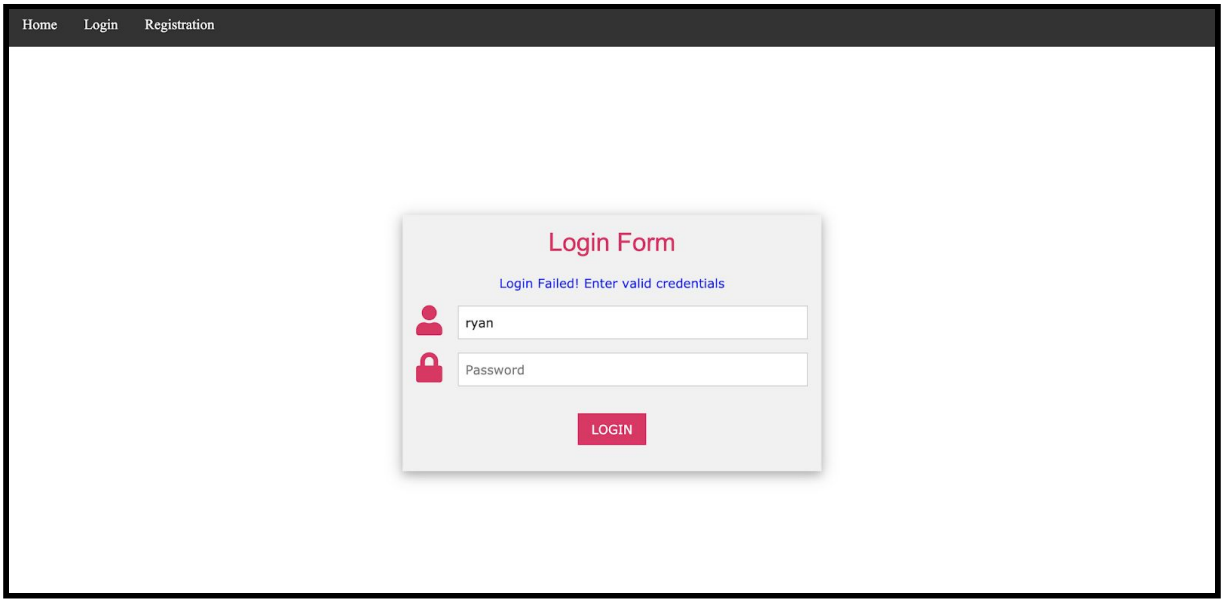


Figure 6:Login Failed

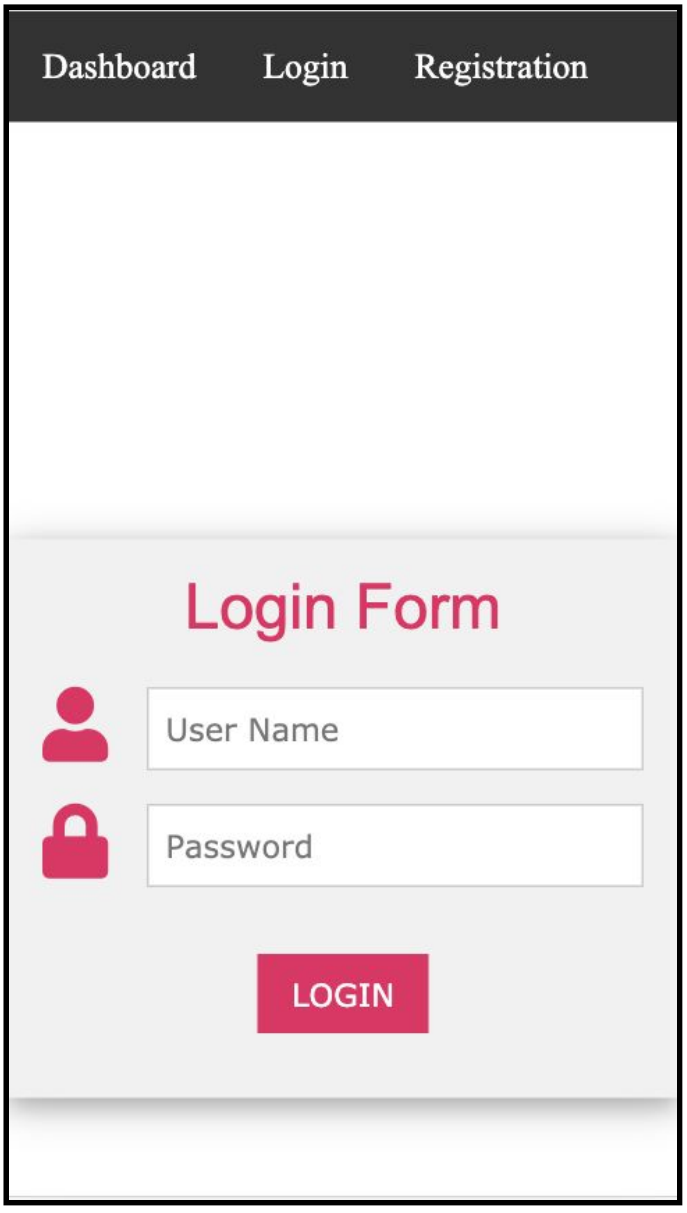


Figure 7:Responsive Login Page

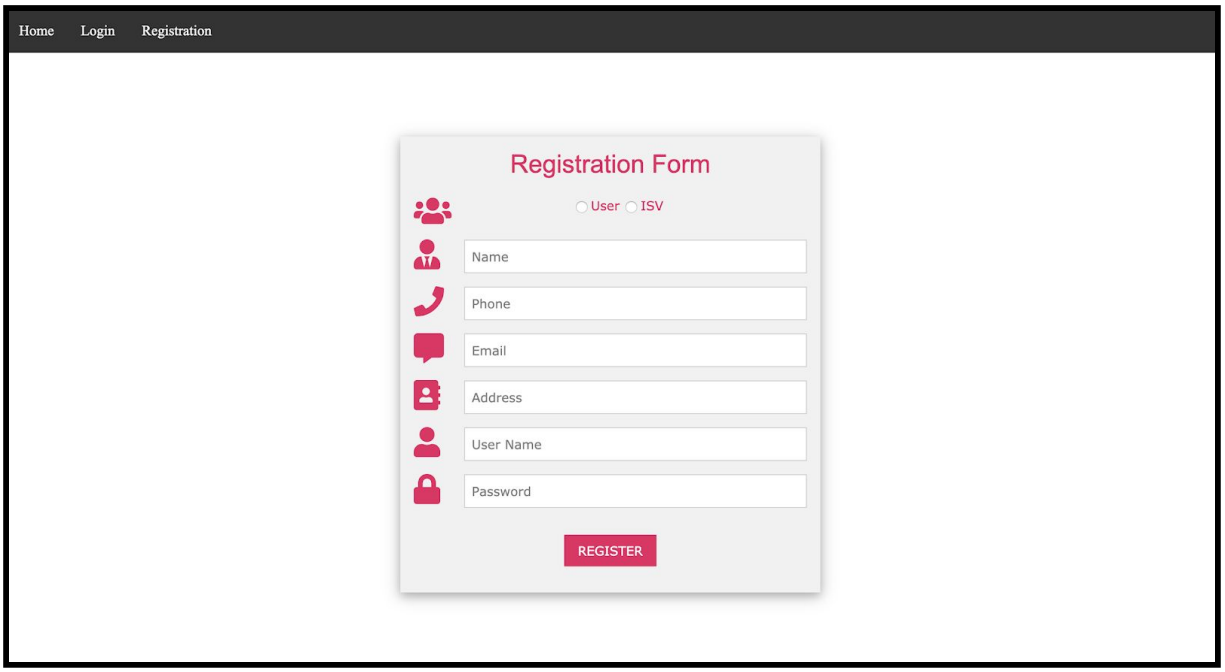


Figure 8:Registration Form

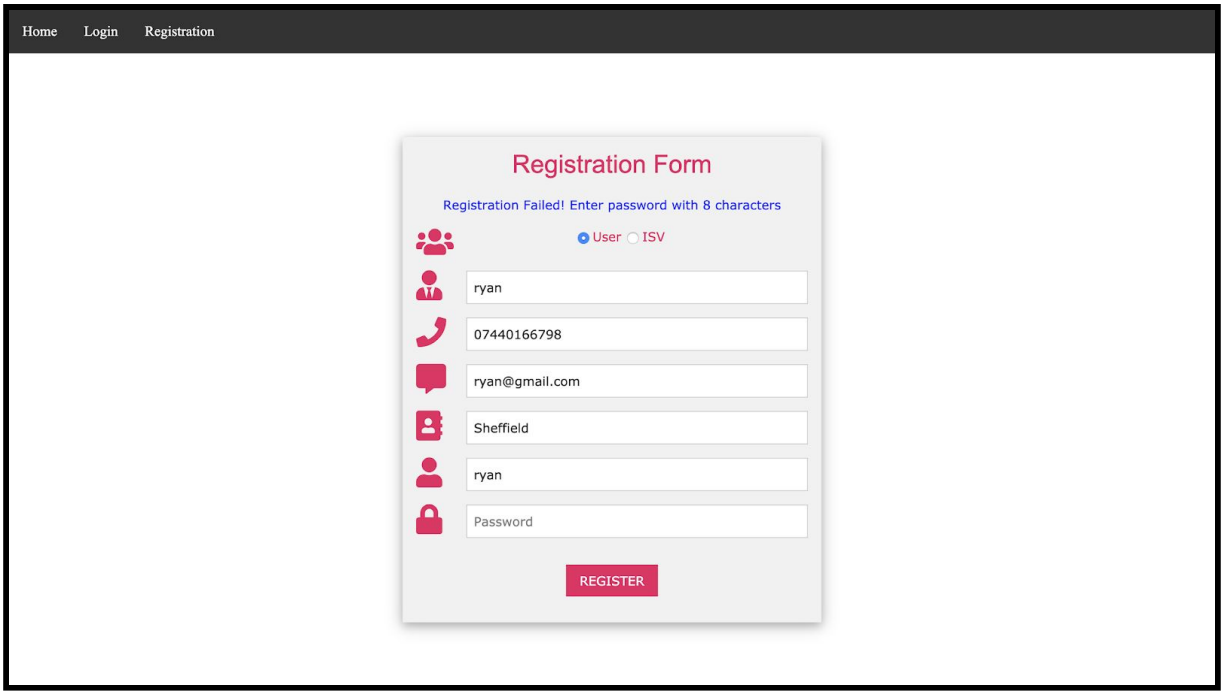


Figure 9:Registration Failed

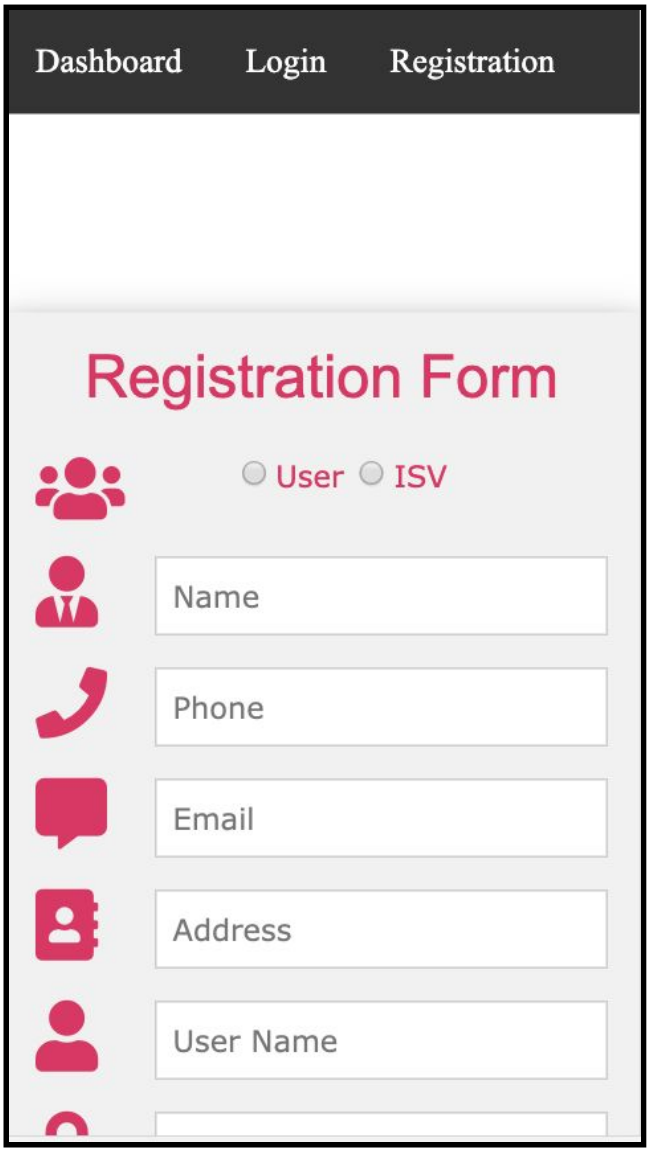


Figure 10:Responsive Registration Page

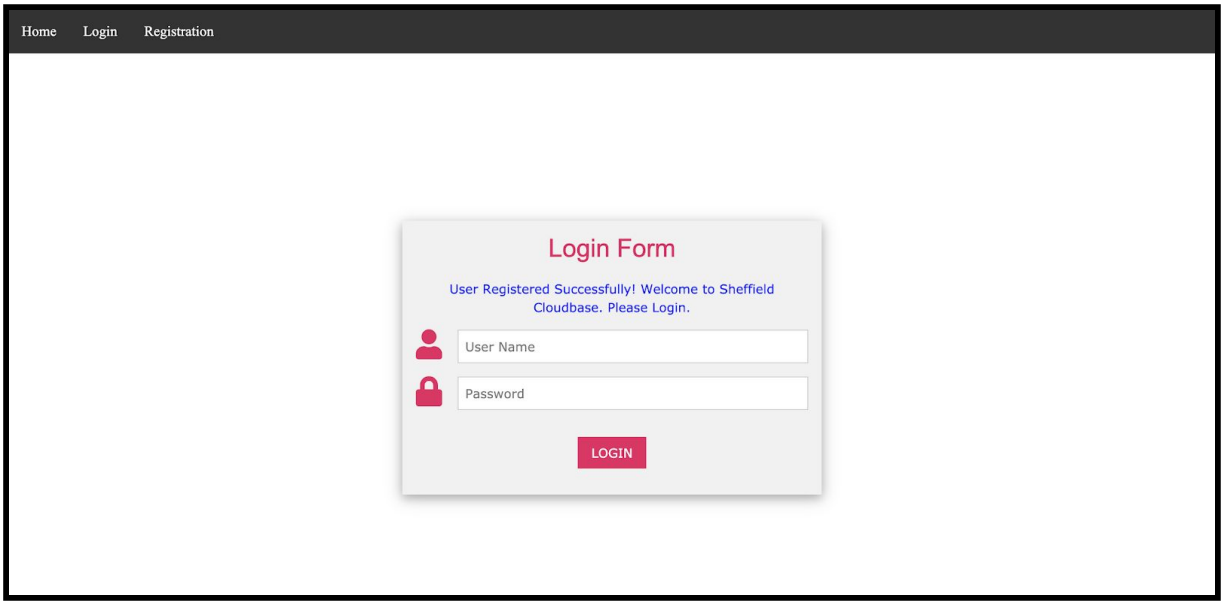


Figure 11:After Successful Registration

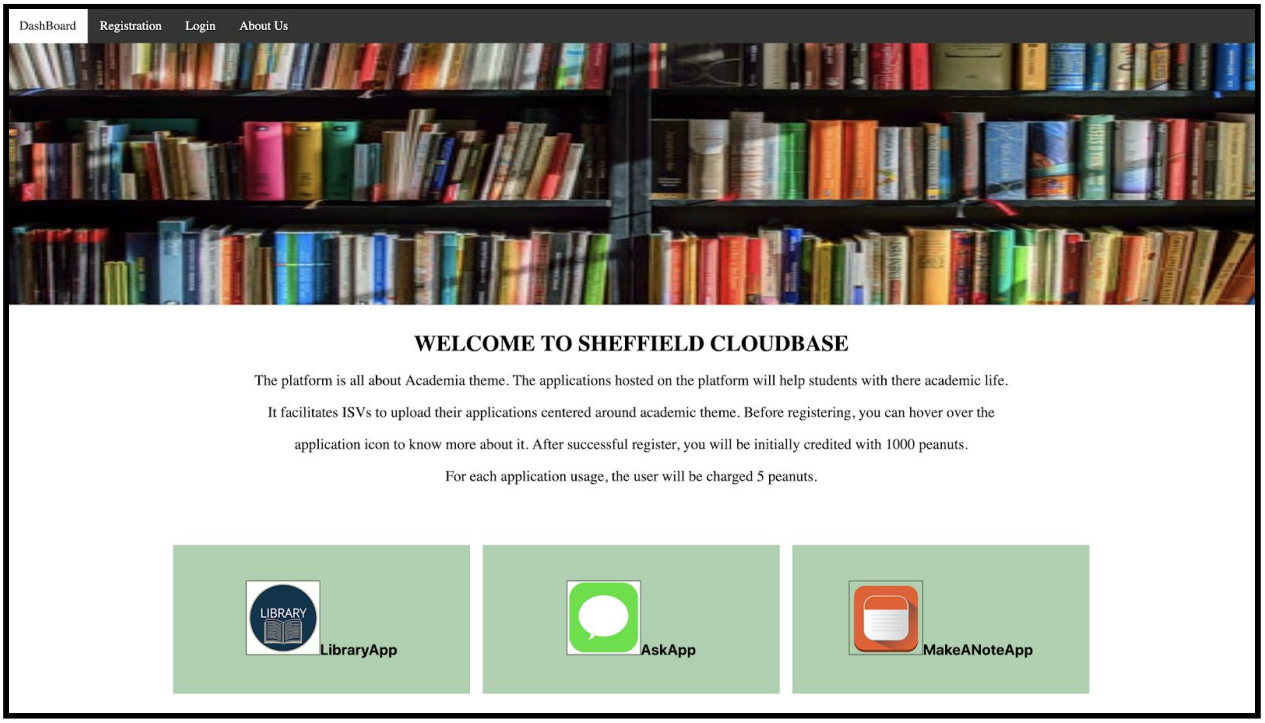


Figure 12:Main Dashboard

Figure 13:Main Dashboard- Responsive

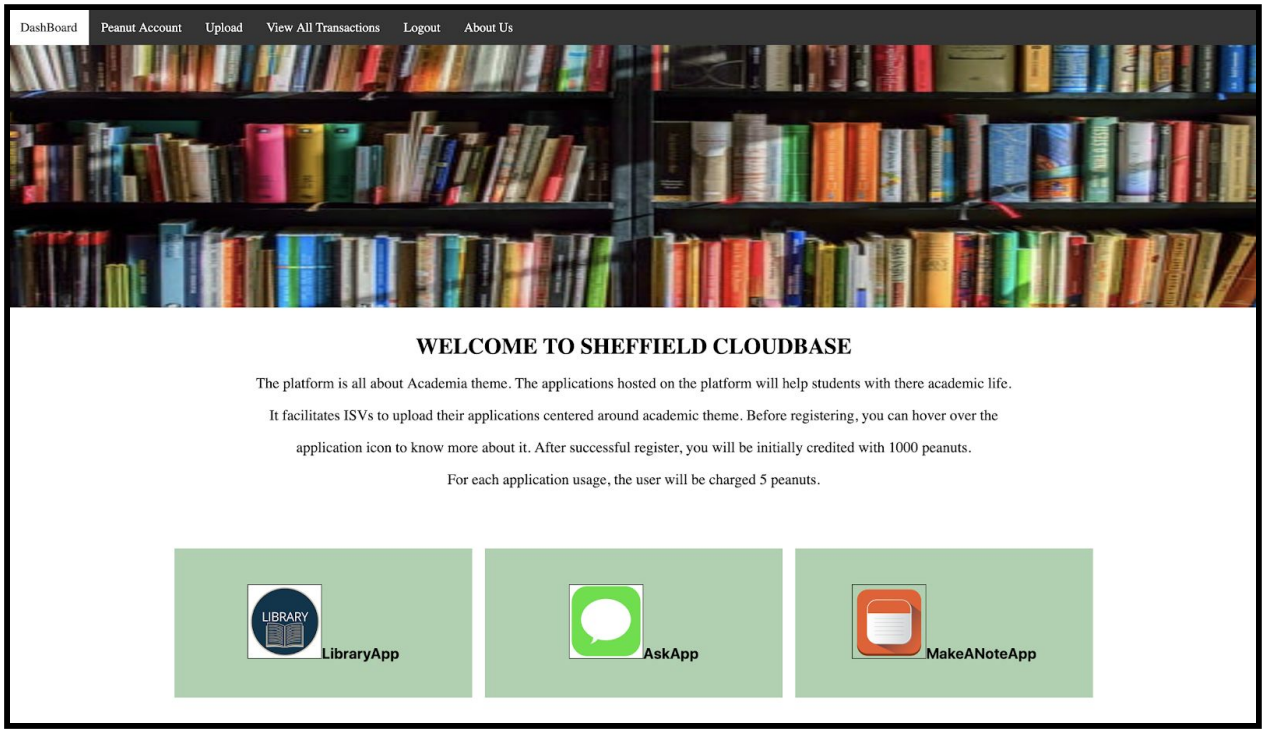


Figure 14:Admin Dashboard

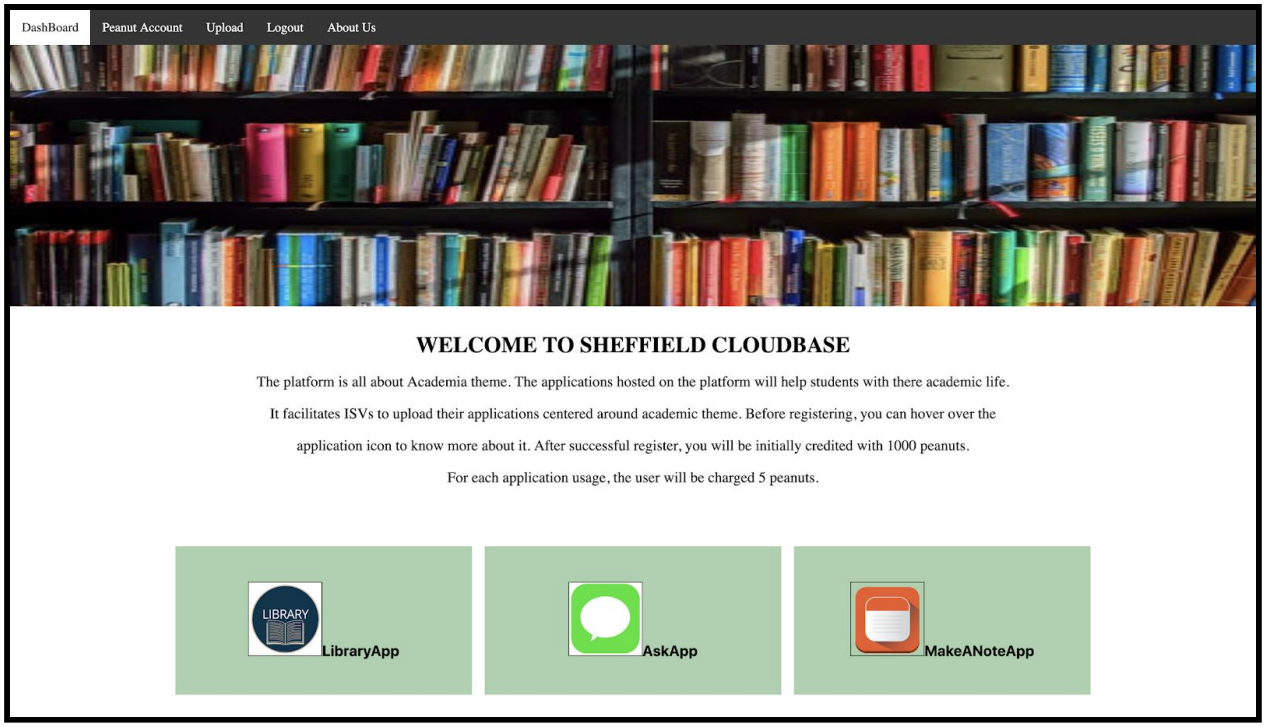


Figure 15:ISV Dashboard

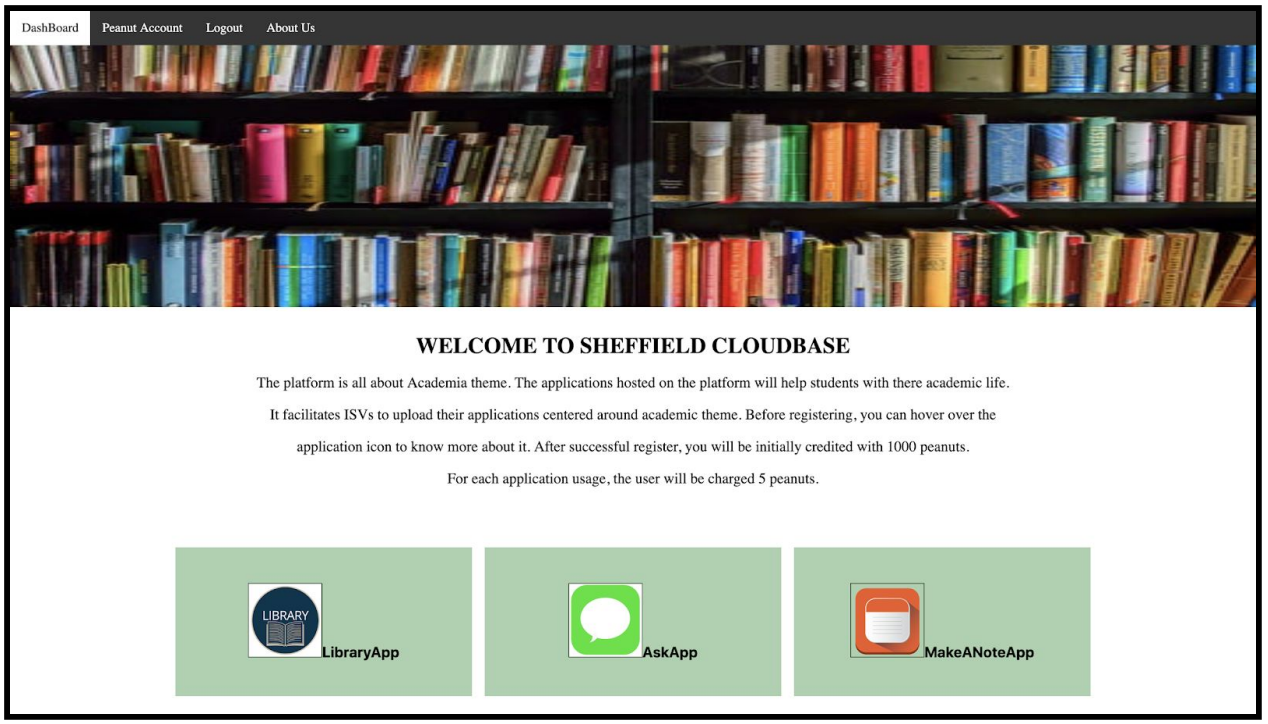


Figure 16:User Dashboard

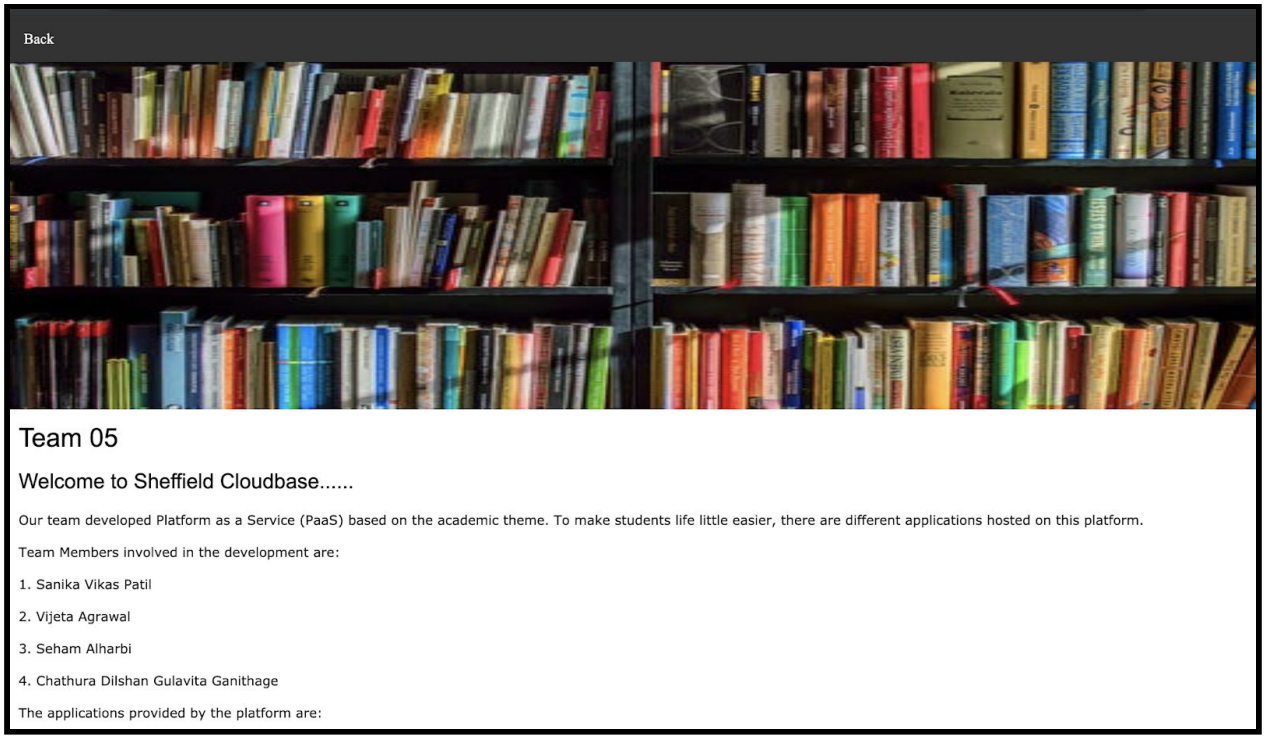


Figure 17:About Page

4.2 Payment microservice

The Payment microservice is also implemented using Spring MVC framework. Figure 16. shows the architecture used to develop this microservice. MVC (Model-View-Controller) architecture has centralized navigation control, which is handled by the Controller. All the incoming request are intercepted by the DispatcherServlet that works as the front controller. In Spring MVC, the model (business entities), controllers (business logic), and views (presentation logic) lie in logical/physical layers, independent of each other. Figure 17 shows the system architecture.

Payment Microserice is using JdbcTemplate for JDBC data access, which presents as an abstraction layer for communication between the microservice and MySQL database server. It uses Restful web service for communicating with Single Sign In/Dashboard microservice. HttpRequest/Response is implemented to communicate with the applications that are developed by the application developers and the ISVs uploads.

At the time of user registration, each user is credited with 1000 peanuts which he/she can use as a currency to use applications on our platform. Their account is created in our Peanut Bank. When they log in through **SingleSignIn** microservice, they can view their balance in their peanut account and the transactions that occurred on their account for each app usage. When they click on an application button, they are redirected to the checkout page before they can use the application. Once the transaction is completed, 5 peanuts are deducted from their account and credited to the accounts of application and platform developers. If the transaction is successful, the user is redirected to the application home page. The Peanut Bank and user peanut accounts are managed by the Payment Microservice.

Connection pooling is done using DataSource objects. The connections that are produced by instances of the DataSource class will automatically be pooled connections.

Instead of creating new connections during every web request, the connections will be provided by the pool of database connections, to improve performance. In Payment microservice, connection pooling is handled by Java Database Connectivity (JDBC) API and the Apache DBCP pooling library.

Both SingleSignIn and Payment Microservices are following the same architecture and guidelines as presented during product pitch phase, except for communication with applications. The **Microservice** database is shared by the both microservices and the **common_db** database is shared by all the services and applications on the platform for storing and retrieving session information. Figure 22 and 23 shows the E-R diagram for both the databases. The screenshots for the payment service and its its responsive design are added below.

The code for Payment Microservice has been checked by team member -Sanika Patil, to make sure that is meets the code artefacts and also is readable to other developers.

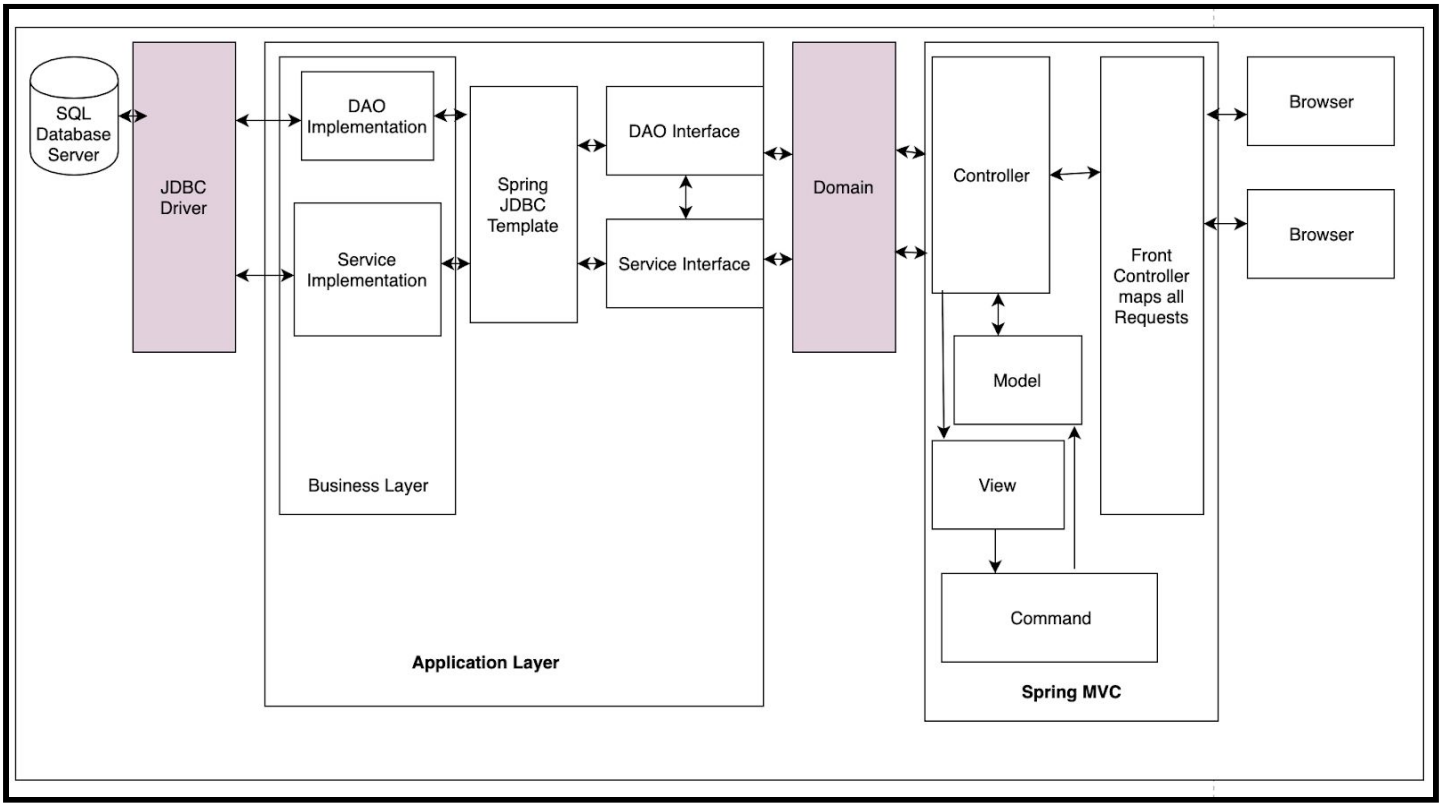


Figure 18:Payment Microservice Architecture

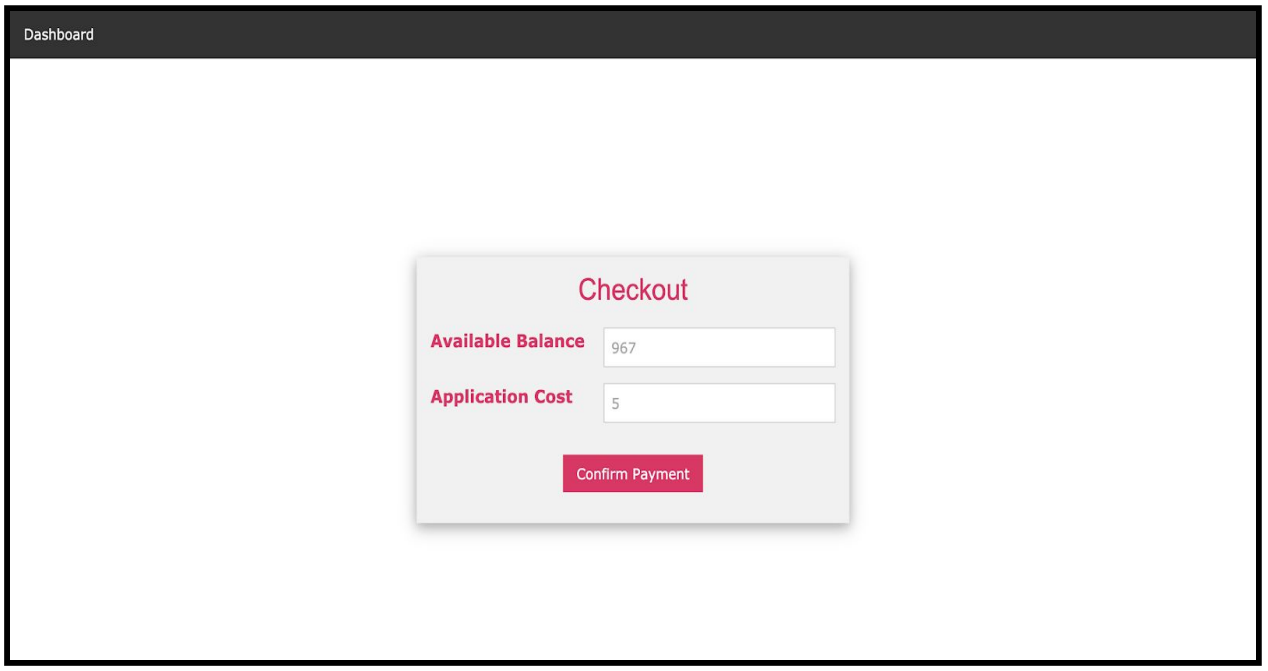


Figure 19:Checkout Page

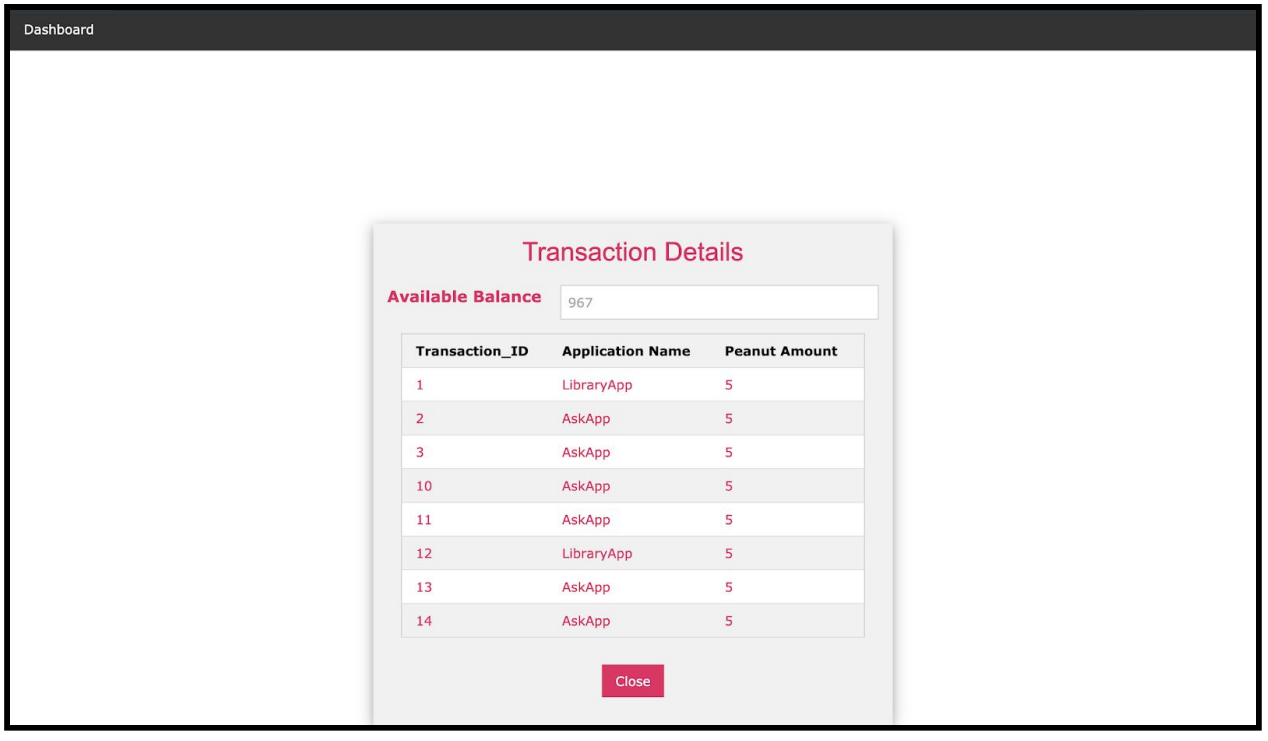


Figure 20:Account Page

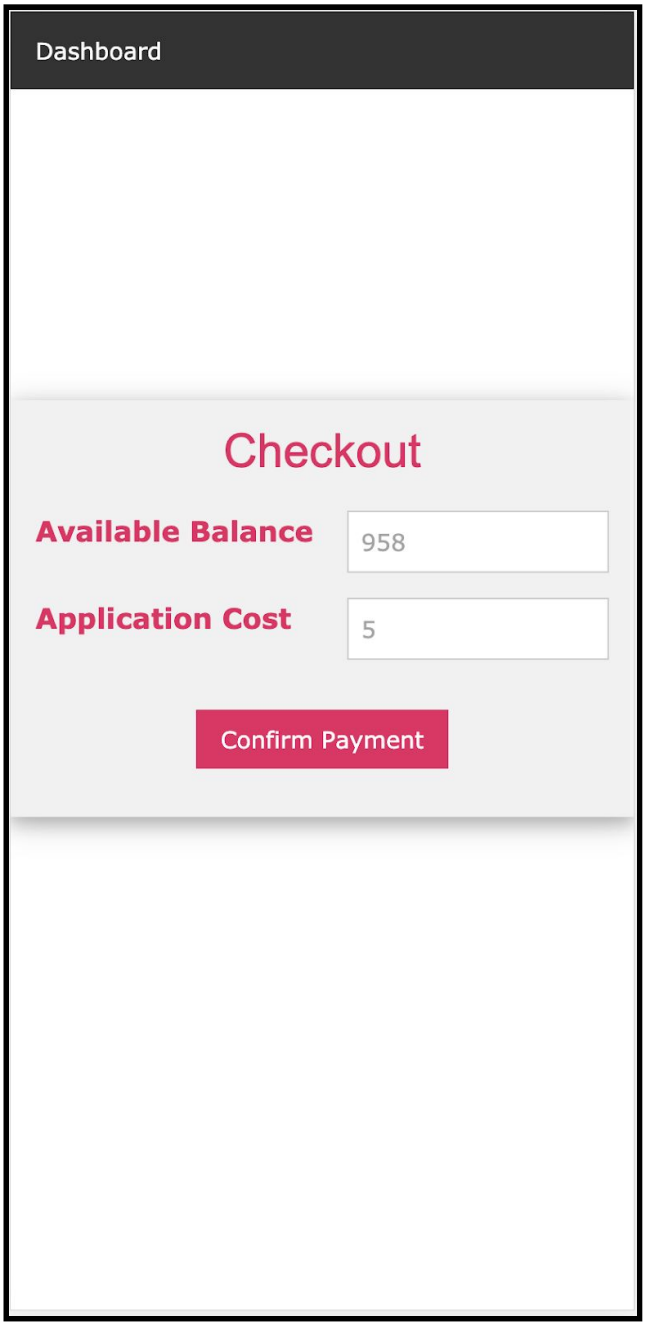


Figure 21:Responsive Checkout Page

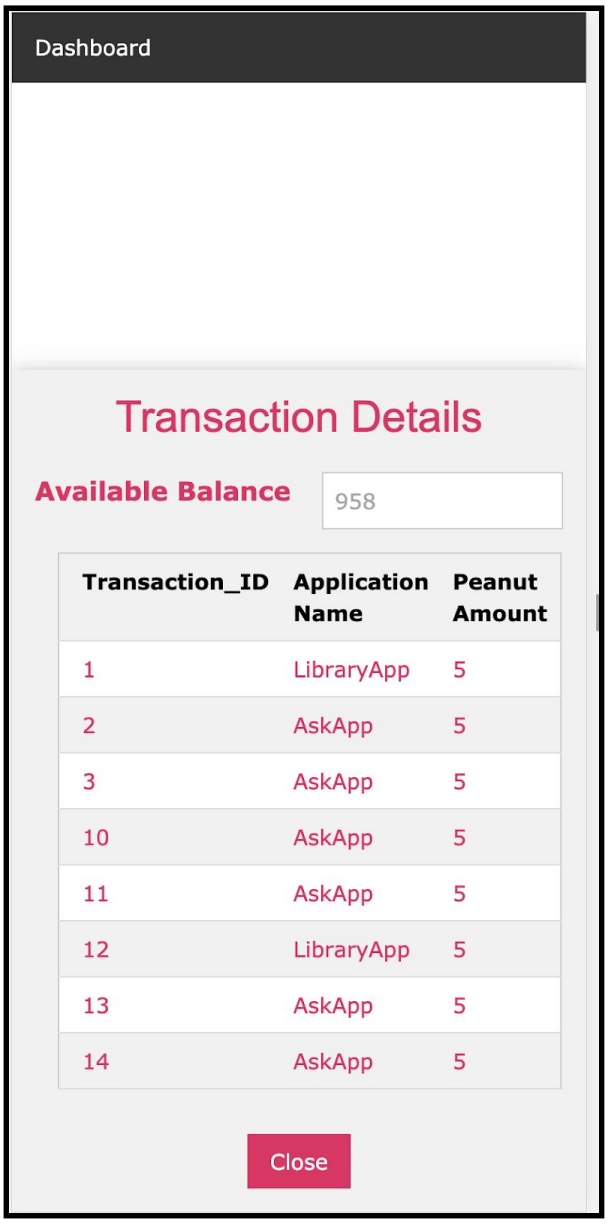


Figure 22:Responsive Account Page

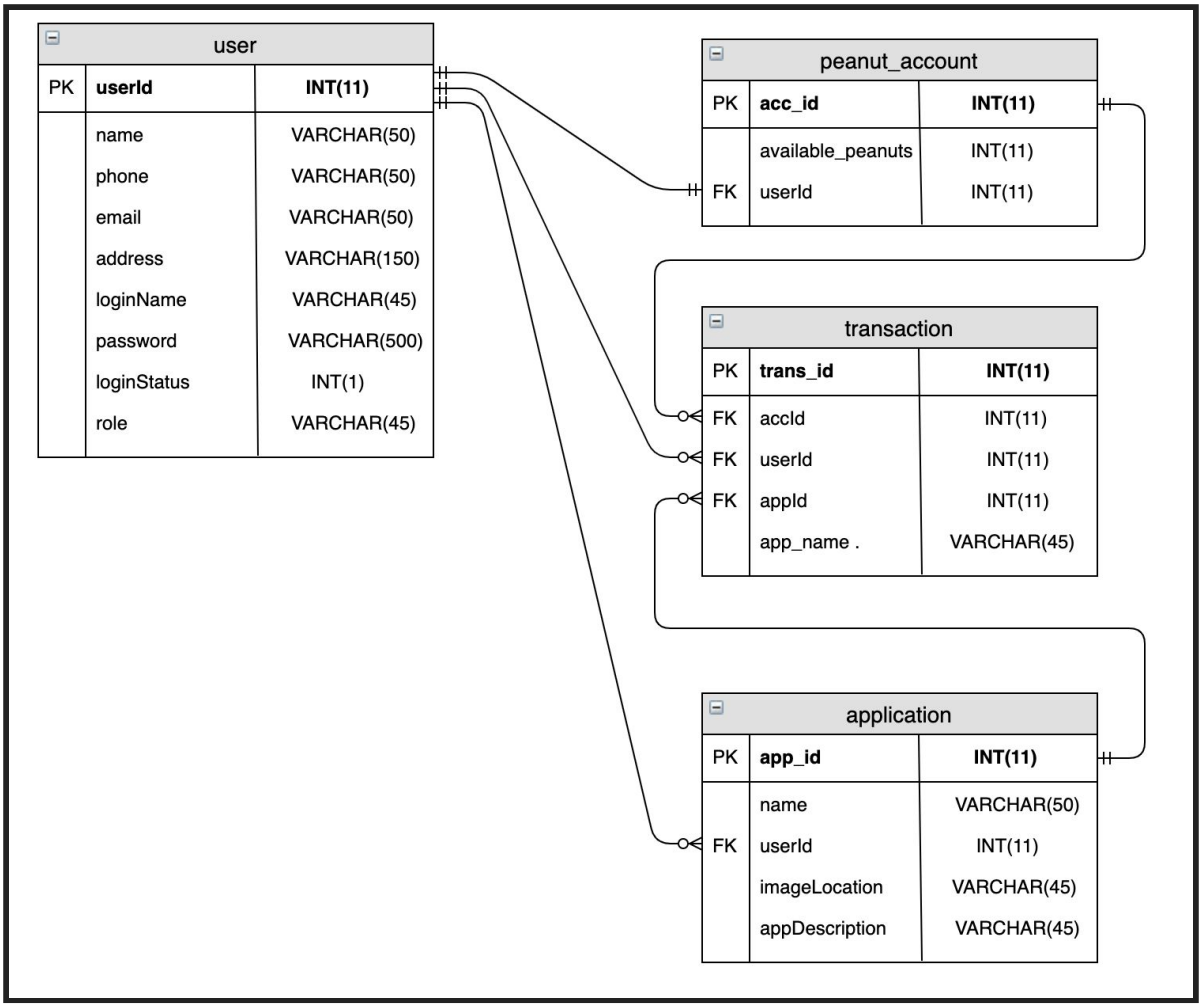


Figure 23:E-R Diagram for Microservice Database

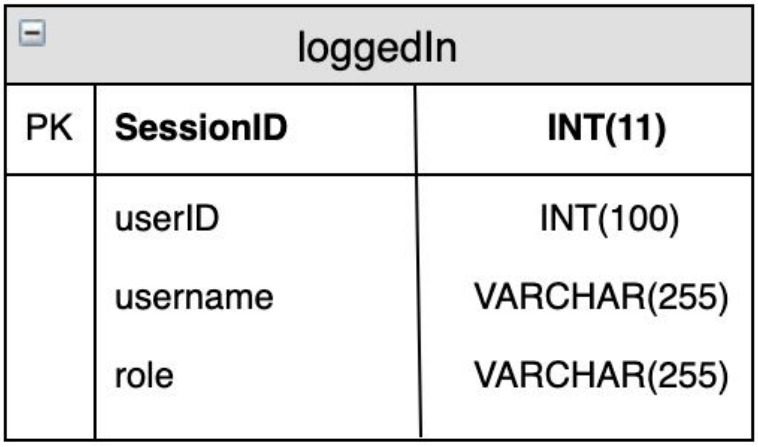


Figure 24:E-R Diagram for common_db Database

4.2.1 Technology Stack used for both Microservices and Dashboard:

- Java SE
- Java EE - JSP, Servlet
- Spring Web MVC
- MYSQL 5.7.26
- Apache Tomcat 8.x
- HTML, W3CSS, JavaScript
- Apache Maven
- Restful Services
- Http Requests

4.2.2 Guidelines used during development of microservices and dashboard:

- Modular Development
    - Web Layer (MVC)
    - Business Layer (Service, DAO)
- Reusability
- Loose Coupling
- Extensibility
- Customizable
- Source code documentation

4.3 Library application

Library application can be considered as a helper app for the students to know what books they have in the university library and their availability and their reviews. As shown in the ER diagram in **Figure 24, "**booktitle", "reviews", "bookcopy" and "reserve" are the tables which are used in the database to facilitate the operations. As shown in **Figure 25** the technologies used are html,css,jsp, java servlets and classes and "mysql" was used as the database technology. Java server pages(jsp) is used to create dynamic web pages and

whenever a form is submitted appropriate servlet is triggered and it calls methods of the relevant java classes. Here servlets act as controllers and java classes are designed to work as the business model. W3 CSS stylesheet was used to obtain an attractive interface.In order to reduce code repetition and to facilitate ease of maintenance “include” method of jsp is used to include the authorisation and navigation bars which are common to every page.

As the user clicks on this app from the “Dashboard” it calls a Servlet called “Init” in the “LibraryApp” domain. Here the header parameters(userID and sessionID) are retrieved and checked with the use of “UserSearch” class from the shared database if the parameters are valid and redirects the user to the index page. Unauthorised users will be redirected to the login page of the platform.

In this application there are two types of users that are considered. Namely they are administrators and normal users. Administrators have the additional privilege to add book information to the application and view details of each book copy whereas users can search books,view book information, reserve books, view reviews and add reviews.

If the user is not logged in, no page is accessible as in every jsp page it checks the session “username” attribute and if it is null the user is redirected to the login page. As some pages are only restricted to the administrators, the session “role” attribute is considered to provide access levels.

When an administrator logs in he/she will be redirected to the “Add new Book” page and there information about the books should be added and the form will call the “AddBookCheck” servlet and with the use of “addBooks“ method of “DBConnection” class the data is inserted to the database. This also creates another table to keep track of each book copy.

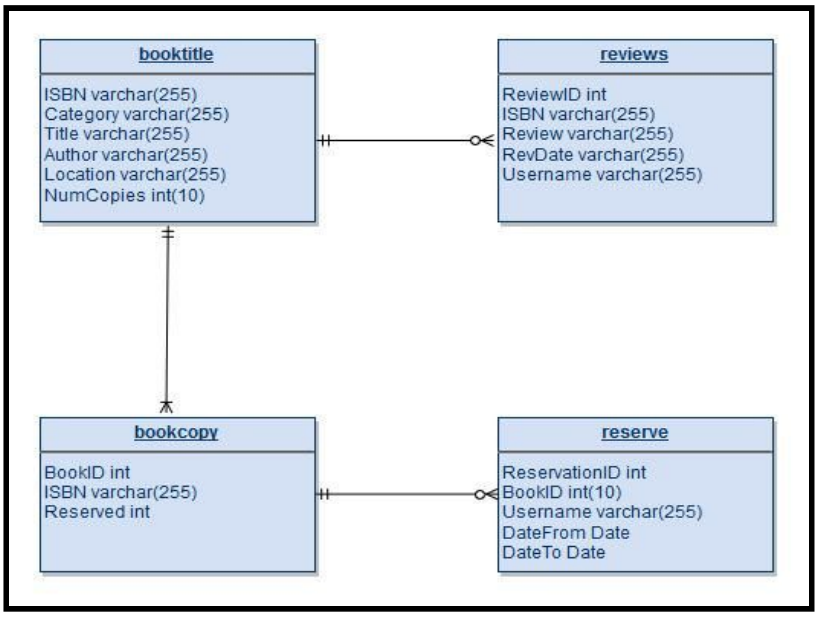


Figure 25:Library application-ER Diagram

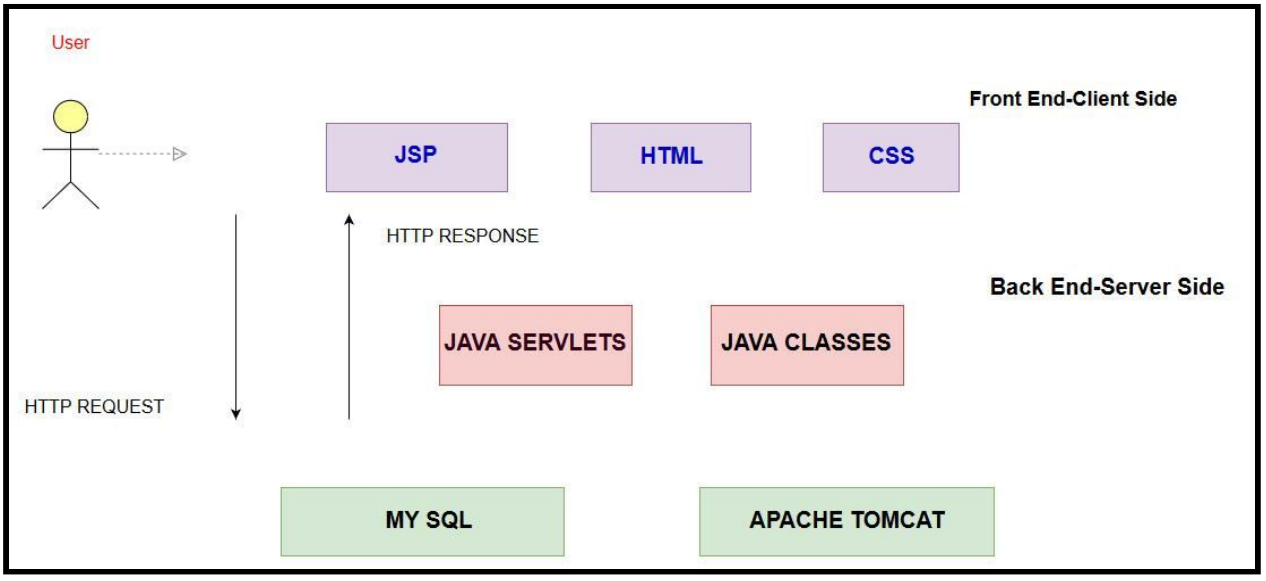


Figure 26:Library application-Technology Stack Diagram

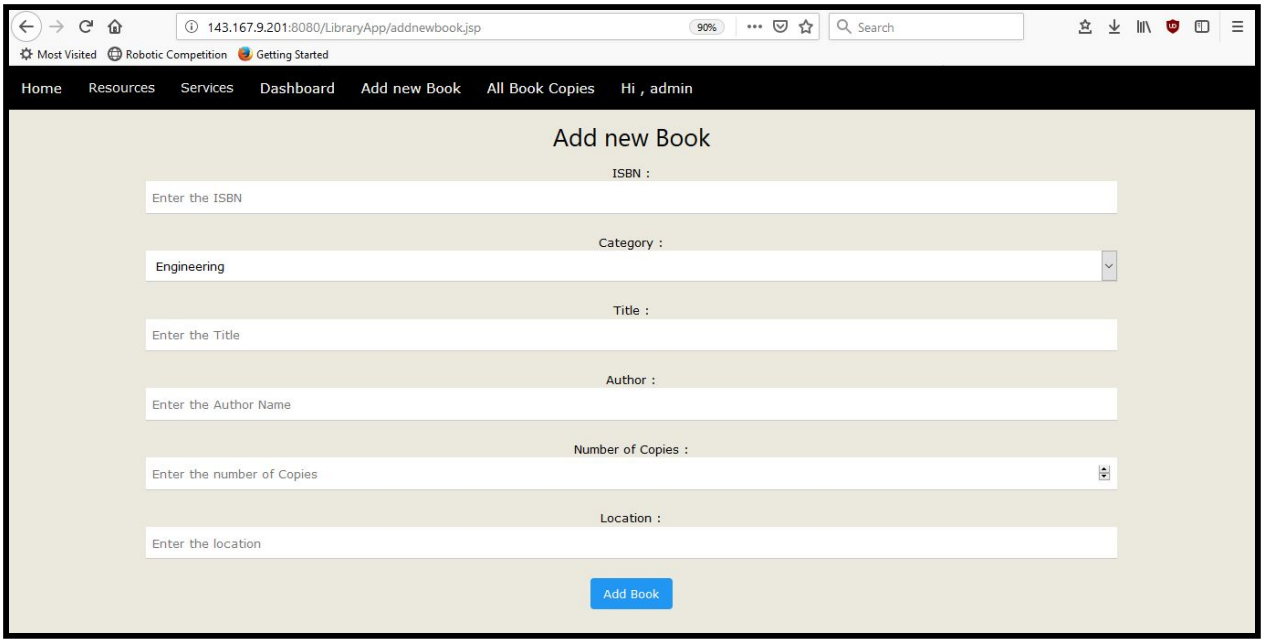
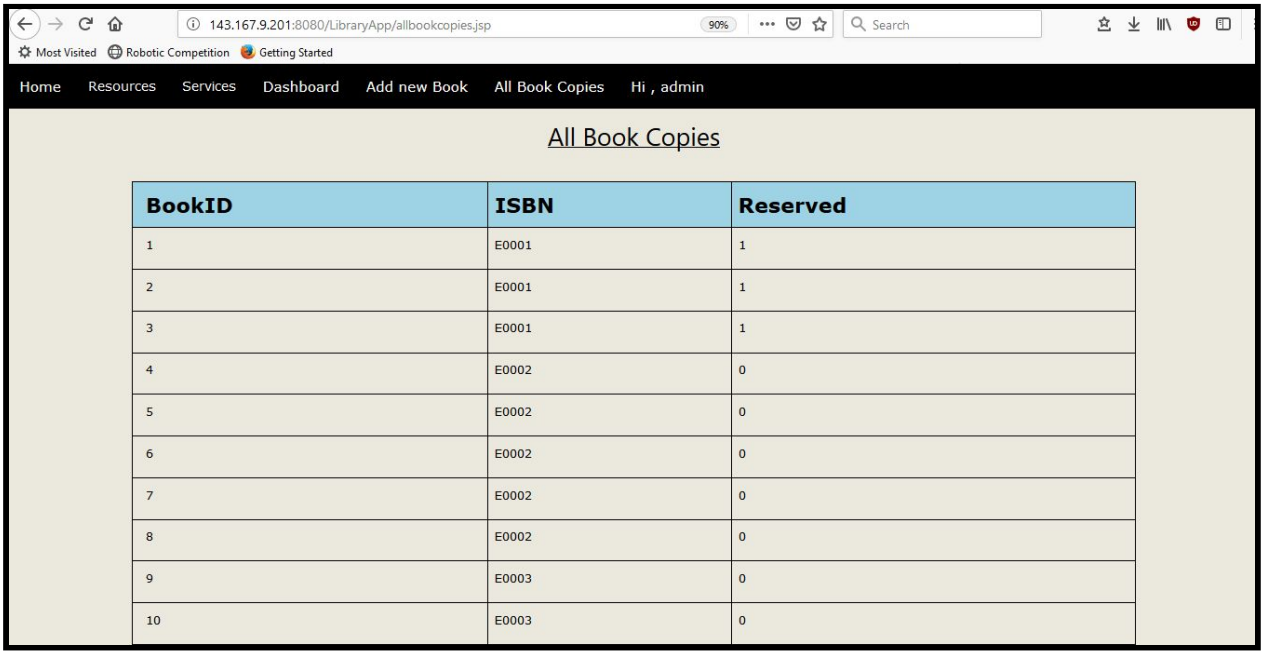


Figure 27:Library application-Add new Book page



BookID	ISBN	Reserved
1	E0001	1
2	E0001	1
3	E0001	1
4	E0002	0
5	E0002	0
6	E0002	0
7	E0002	0
8	E0002	0
9	E0003	0
10	E0003	0

Figure 28:Library application-All Book Copies

When logged in, the user will be redirected to the home page(index.jsp) as shown in **Figure 28**. Also as a main requirement was to develop a responsive web design **Figure 29** is to demonstrate the view when size of the screen changed. There the user can search books. Once the search button is pressed it will go to search results page(**Figure 30**) which will give the option to choose the appropriate result. Here to find results from the database “LIKE” keyword is used.

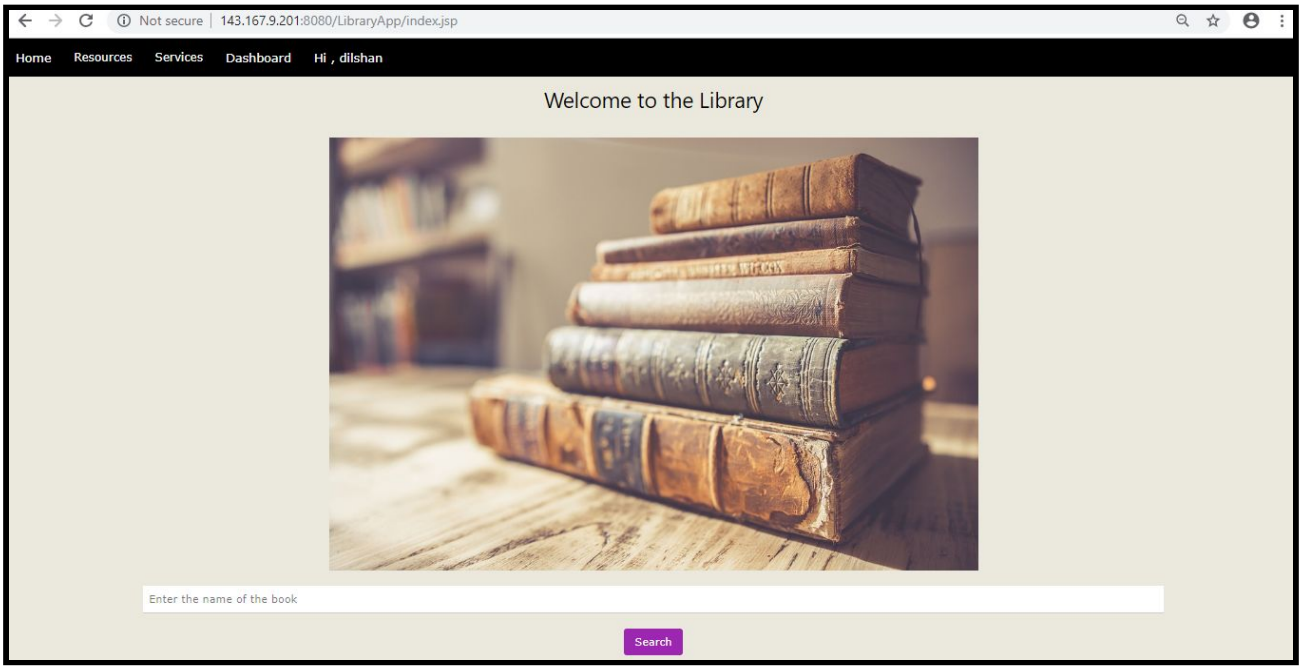


Figure 29:Library application-Home page(for users)

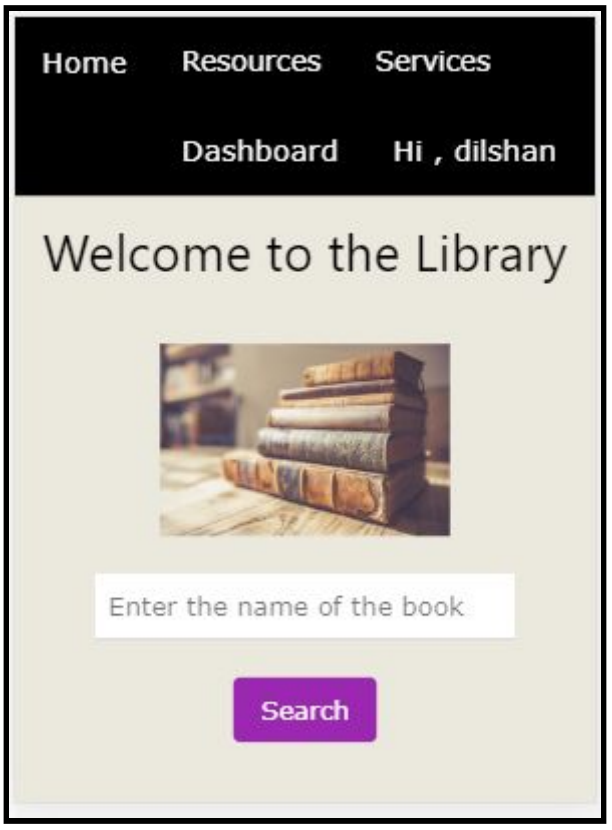


Figure 30:Library application-Home page(responsive design)

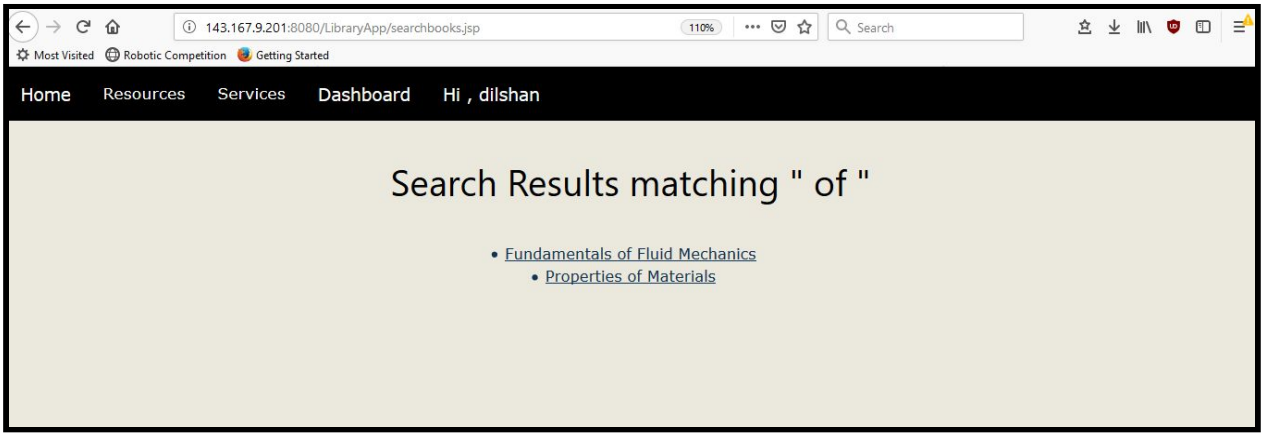


Figure 31:Library application-Search Results

As shown in **Figure 31** when the user selects a book he/she can view all the information as the category, author name, ISBN, location of the book, number of copies and availability. Moreover, reviews of the book also can be seen. User has the privilege to reserve a copy or add a review.

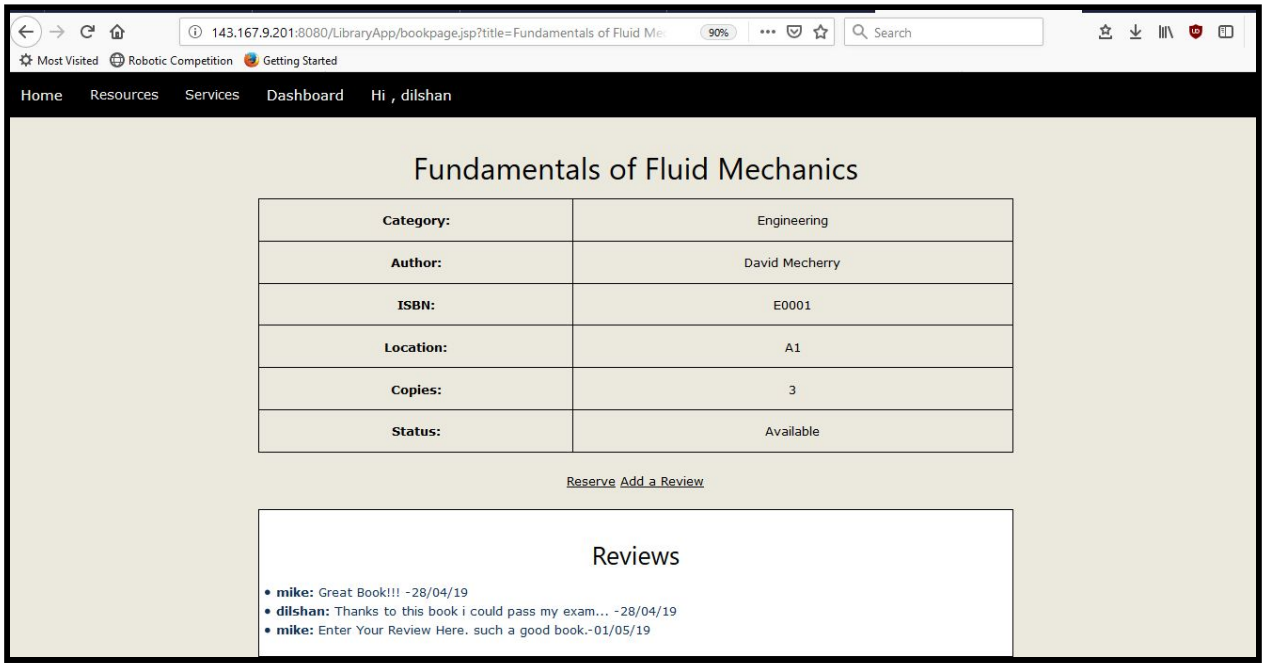


Figure 32:Library application-Book information

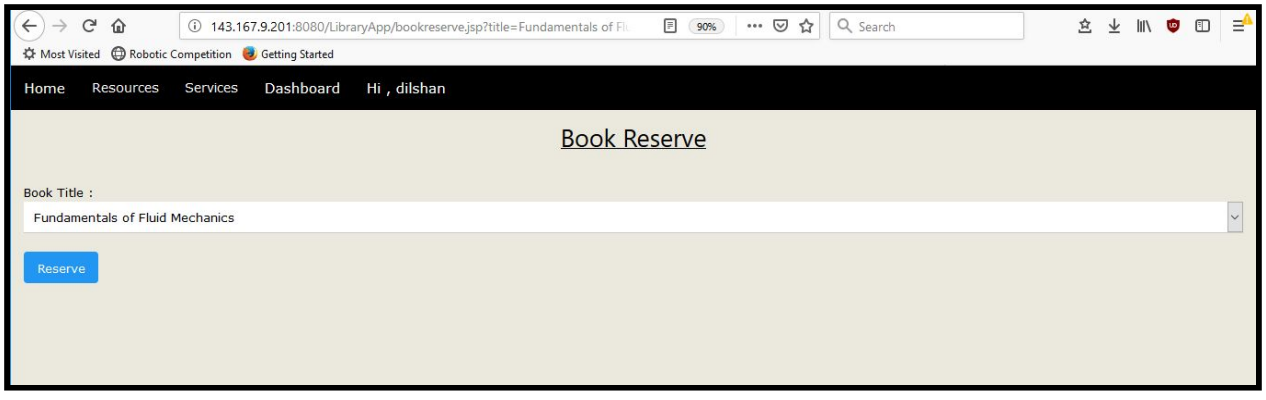


Figure 33:Library application-Book reserve

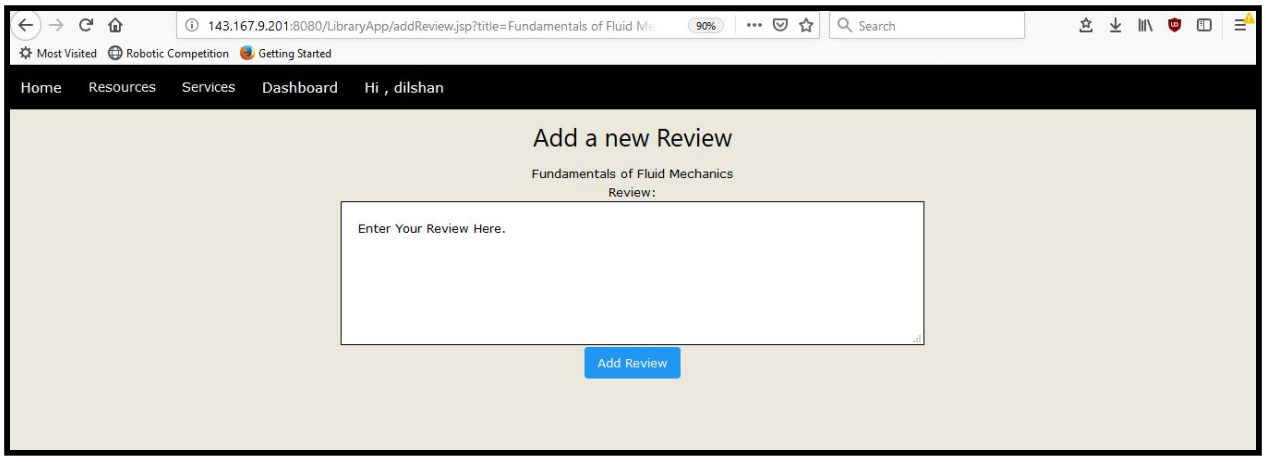
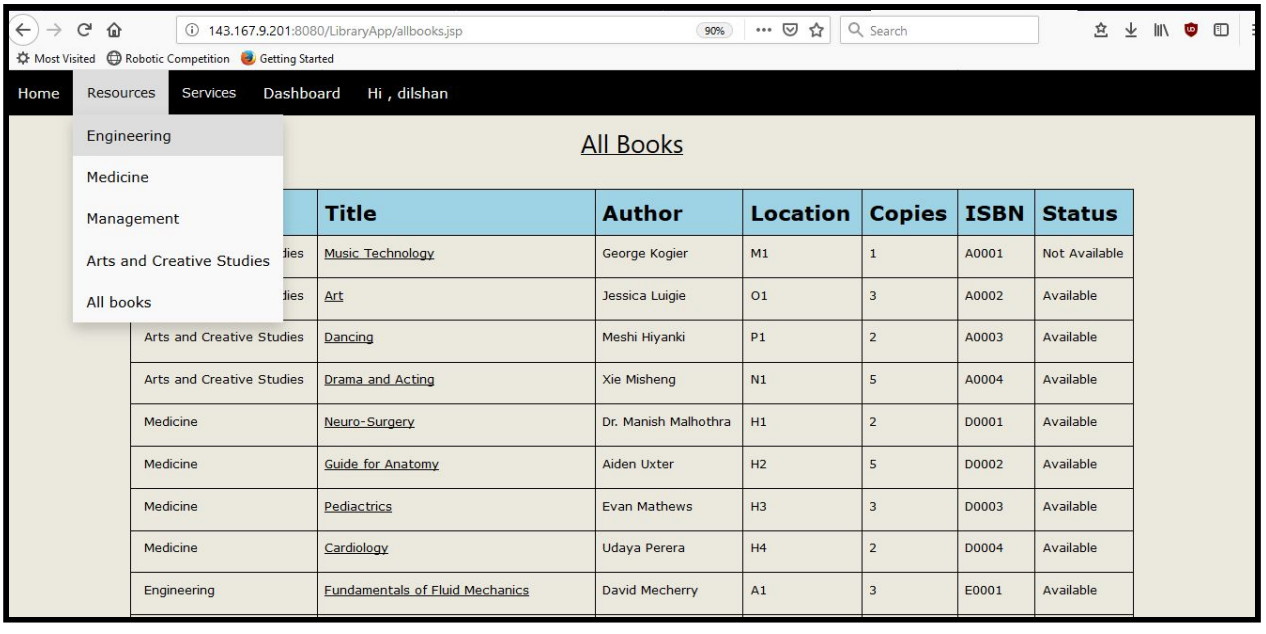


Figure 34:Library application- Add new Review

In the top menu user can view the resources by filtering by category or view all books. As shown in the **Figure 34** the user can view the book information and the status too.



	Title	Author	Location	Copies	ISBN	Status
	Music Technology	George Kogier	M1	1	A0001	Not Available
	Art	Jessica Luigie	O1	3	A0002	Available
Arts and Creative Studies	Dancing	Meshi Hiyanki	P1	2	A0003	Available
Arts and Creative Studies	Drama and Acting	Xie Misheng	N1	5	A0004	Available
Medicine	Neuro-Surgery	Dr. Manish Malhothra	H1	2	D0001	Available
Medicine	Guide for Anatomy	Aiden Uxter	H2	5	D0002	Available
Medicine	Pediactrics	Evan Mathews	H3	3	D0003	Available
Medicine	Cardiology	Udaya Perera	H4	2	D0004	Available
Engineering	Fundamentals of Fluid Mechanics	David Mecherry	A1	3	E0001	Available

Figure 35:Library application- All books

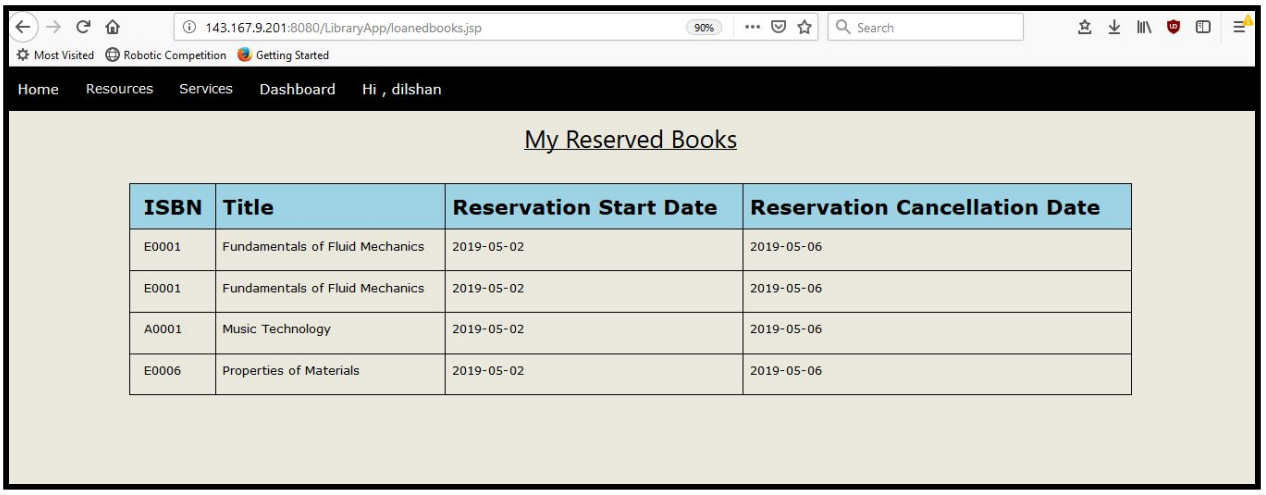


Figure 36:Library application- Users reserved books

As shown in **Figure 35** the user can view the current reserved books and the date the reservation gets cancelled. For this operation every jsp page includes “reservecancellation.jsp” which checks the current date and cancels any reservation which is expired. This will be updated in “reserve” and “bookcopy” tables.

```
mysql> SELECT * FROM reviews;
+----------+-------+-----------------------------------------------+----------+-----------+
| ReviewID | ISBN  | Review                                        | RevDate  | Username  |
+----------+-------+-----------------------------------------------+----------+-----------+
|        1 | E0001 | Great Book!!!
                           | 28/04/19 | mike      |
|        2 | E0001 | Thanks to this book i could pass my exam...
 | 28/04/19 | dilshan   |
|        3 | E0004 | ASD
                                      | 30/04/19 | mike      |
|        4 | E0001 | Enter Your Review Here.
such a good book.    | 01/05/19 | mike      |
|        5 | E0002 | Enter Your Review Here.
good book.           | 01/05/19 | mike      |
|        6 | E0002 | Great book for the ones who love aircrafts
  | 02/05/19 | dilshan   |
|        7 | M0001 | Such a good book.
                        | 02/05/19 | seham     |
|        8 | E0006 | Amazing book!!!
                          | 03/05/19 | tina      |
|        9 | A0001 | this is a good book.                          | 04/05/19 | Sealharbi |
|       10 | D0004 | such a nice book.                             | 04/05/19 | lip17saa  |
+----------+-------+-----------------------------------------------+----------+-----------+
10 rows in set (0.00 sec)

mysql>
```

Figure 37:Library application- Reviews table in the Virtual Machine

As shown in Figure 36, User entries are saved in the database in the virtual machine. The software code of the app was checked and tested by a team member(Seham) and necessary changes were done according to the feedback.

4.4 ASK Application

As it has been mentioned in the business section of this report, Ask application provides an academic platform for the university students to ask questions and to answer other students' questions as well.

The first step that has been taken towards designing the ASK application is to satisfy the device-independent requirement. Therefore, a W3-css stylesheet has been used to make the design of this web application responsive and to make it operates across all major browsers. Figure 37 shows how the application looks like on a desktop browser, and Figure 38 shows how the web design is changing on a mobile phone browser by moving the side navigation menu to a top drop-down list. This responsive web design makes the use of the application more user-friendly and it also helps to display all of its pages appropriately across different devices.

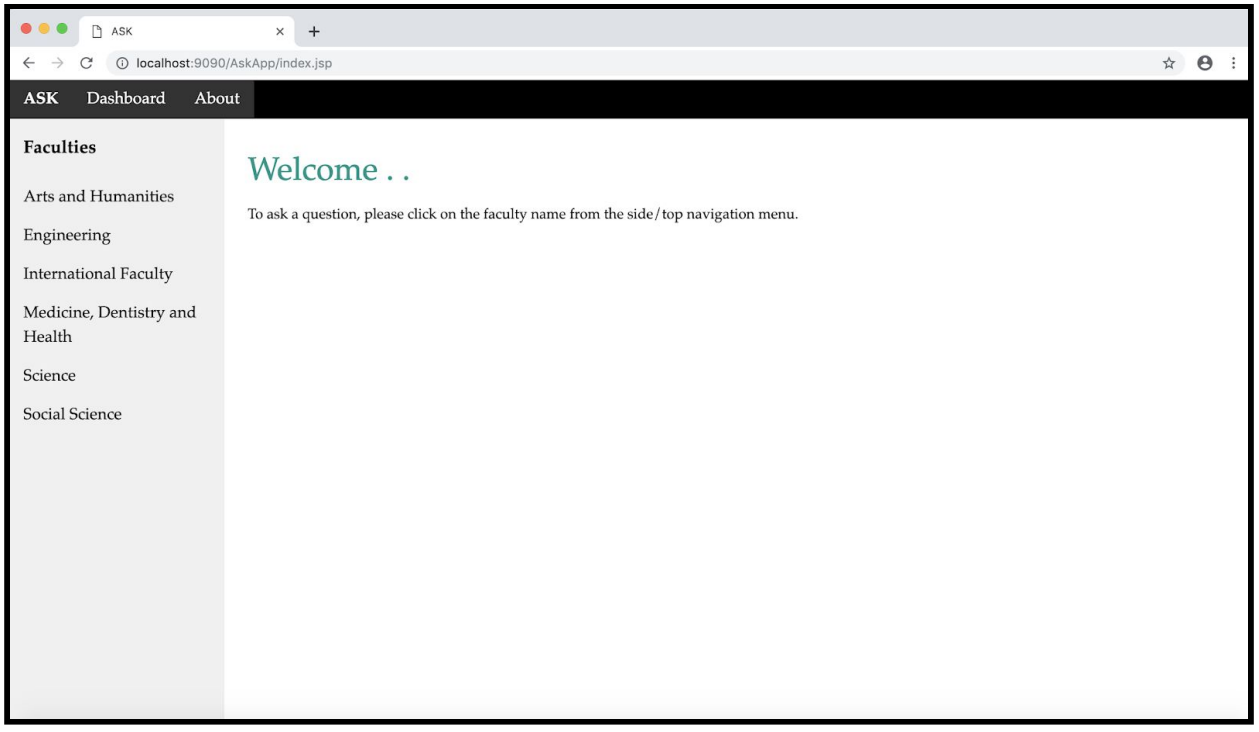


Figure 38:ASK application on a desktop browser.

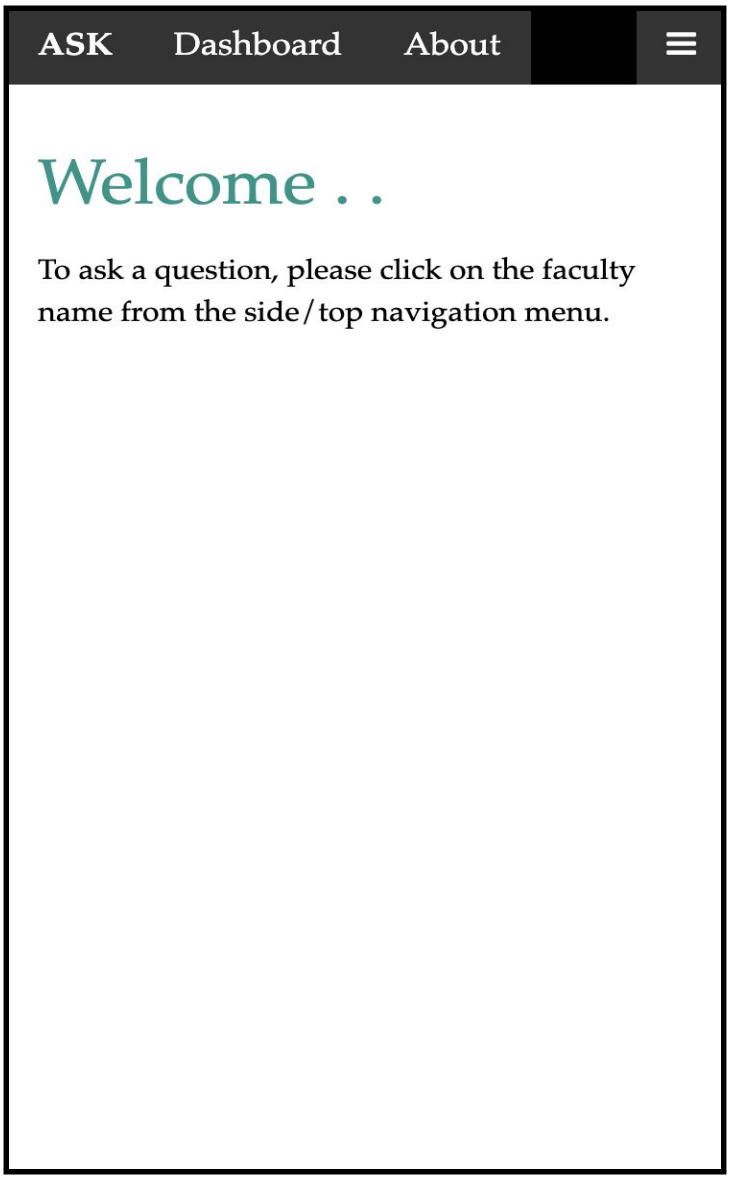


Figure 39:ASK application on a mobile phone browser (Responsive View).

To store the data, ASK application uses MySQL database and the JDBC (Java Database Connectivity) API and its drivers to connect to the database and to allow access to all of its tables. The ERD (Entity Relationship Diagram) of the ASK application's database is shown in Figure 39.

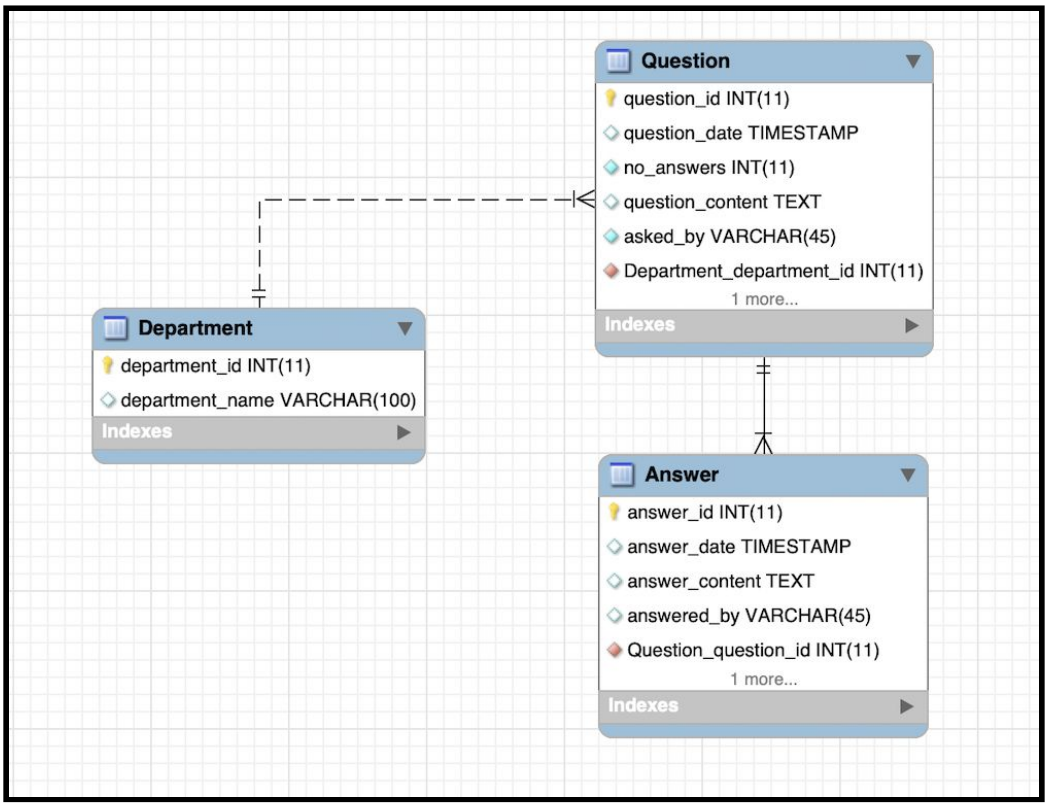


Figure 40:ASK application ERD.

When it comes to the implementation phase of the ASK application, the MVC (Model-View-Controller) architectural pattern, which is shown in Figure 40, has been taken into consideration. Ask application uses a separated java class to represent the Model, and therefore it is the only component that deals with the database and therefore all operations from and to the database are processed under its control.

However, the View component is represented by using JSP pages that contain some HTML5 semantic elements. These JSP pages are dynamically changing depending

on the user interaction with the application. Finally, the Controller component is represented by several Java servlets that receive all user's requests and perform some tasks and then respond to them by updating the view (JSP pages).

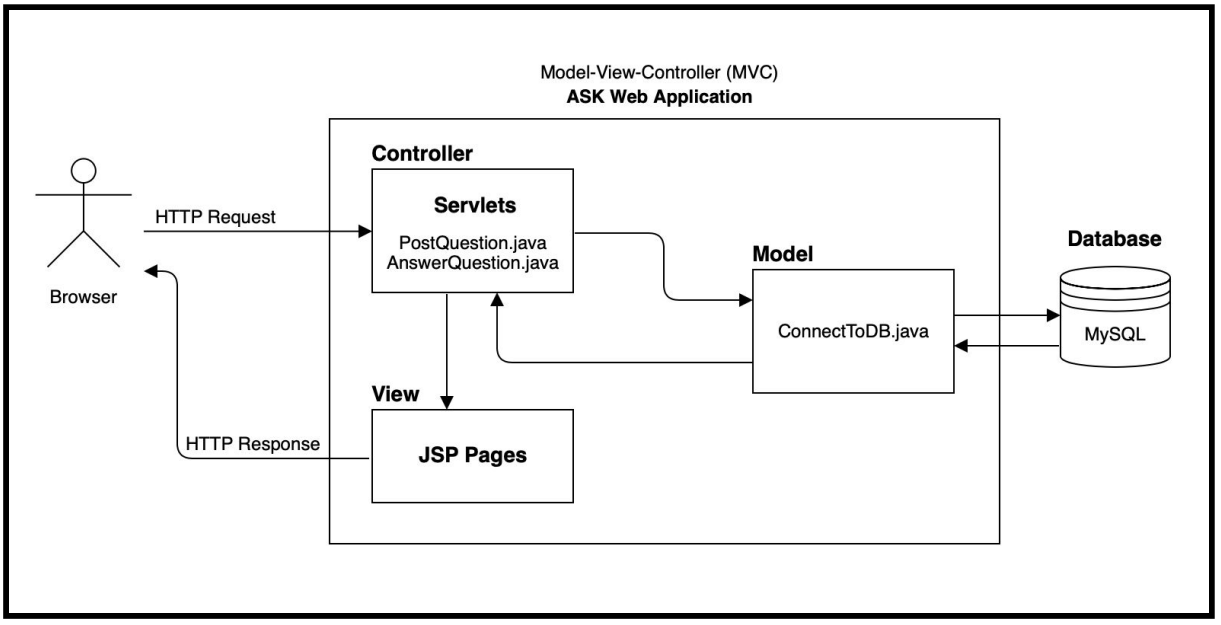


Figure 41:ASK application - MVC Architecture.

In addition, Figure 41 shows all the technologies that have been used to develop Ask application on both the client side and the server side.

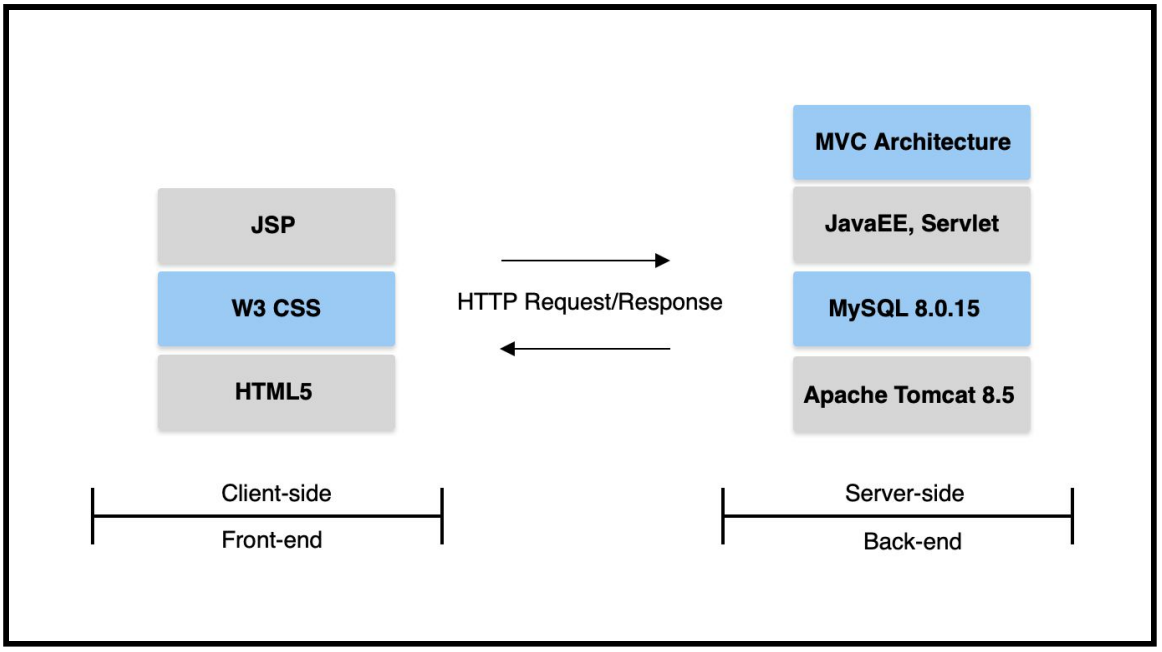


Figure 42:ASK Application - Technology Stack Diagram.

The user of the ASK application is able to posts questions and also reply to other students' questions. These questions and answers are shown along with the user name and the date of the post. In addition, all questions and answers can be of any length; therefore the forms in this web application use a POST method, and this is not just because of the unlimited data length that this method provides compared with the GET method, but also to do not include the question or the answer data in the sent URL.

In addition, all forms trigger appropriate servlets that handle the post of questions and answers. Finally, Figure 42, Figure 43, Figure 44, Figure 45 and Figure 46 show some examples of some pages after deploying and running ASK application on the VM Tomcat server and how these pages look like before, and dynamically change, after answering a particular question in a particular department page (Figure 47 and Figure 48 show how this question and its answer is stored in the back-end data storage).

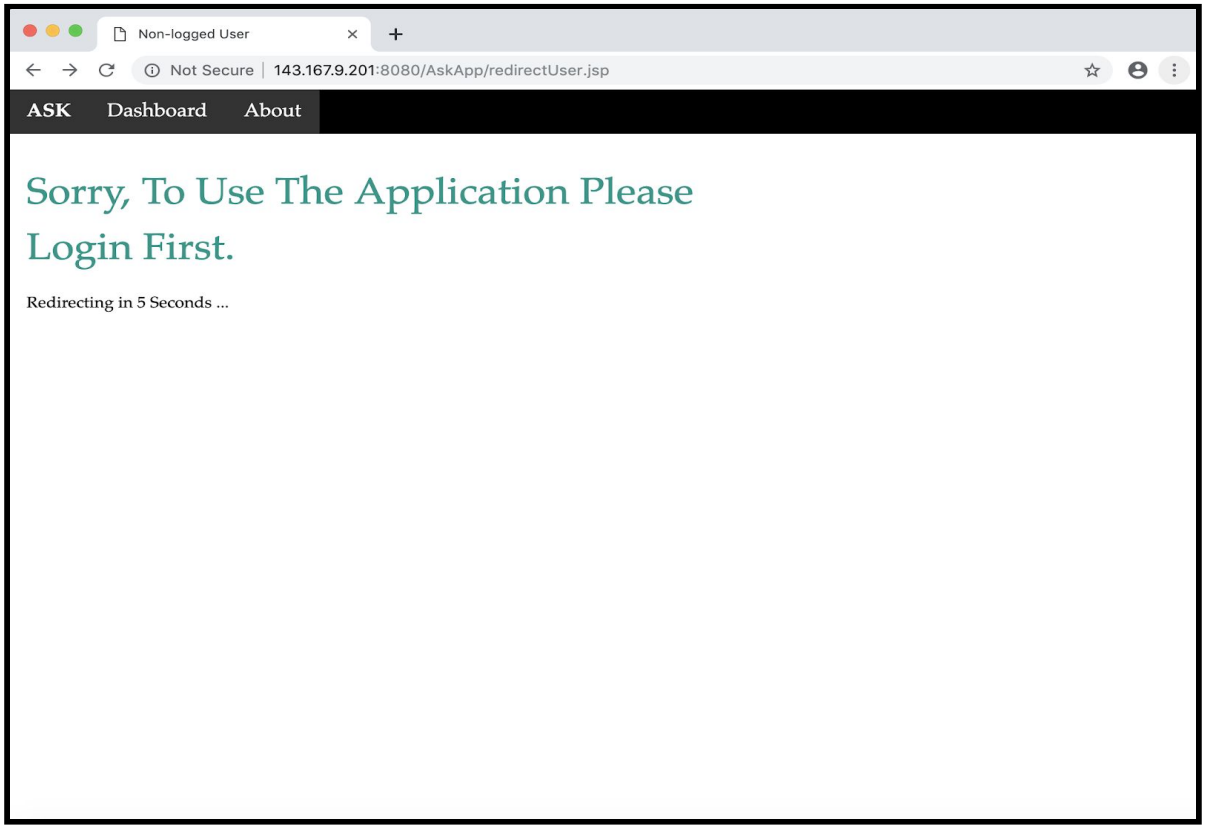


Figure 43:ASK application - redirecting non-logged users.

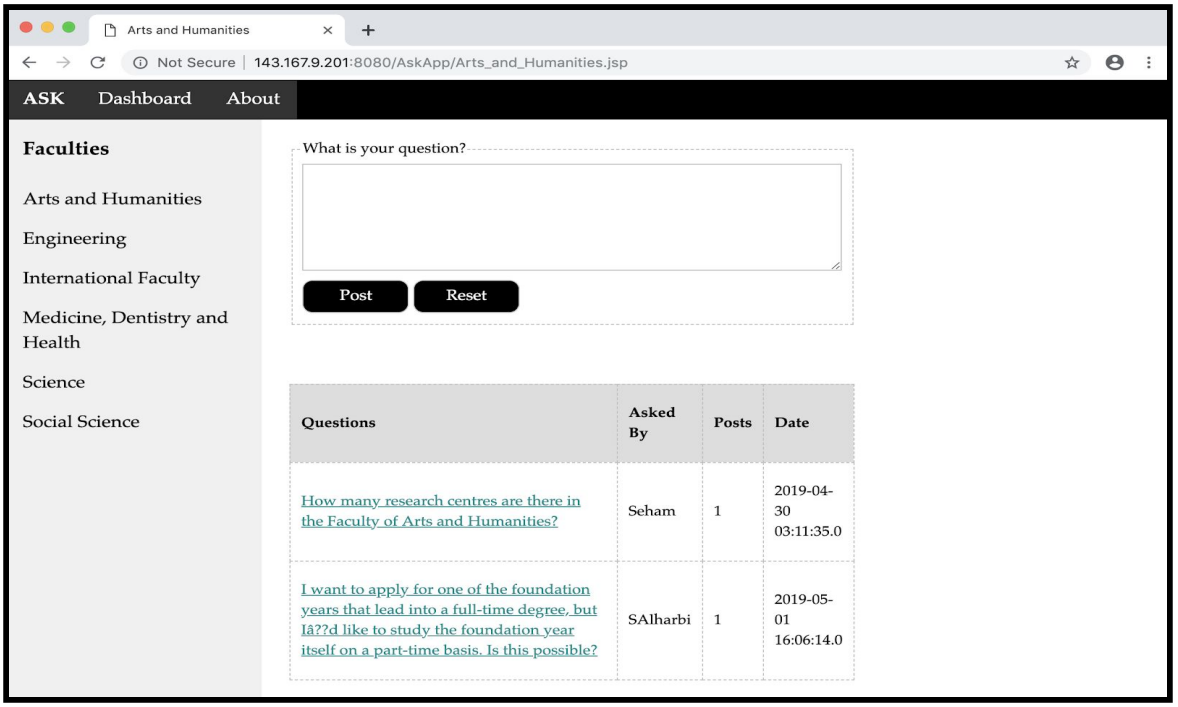


Figure 44:ASK application - one department page with its related questions.

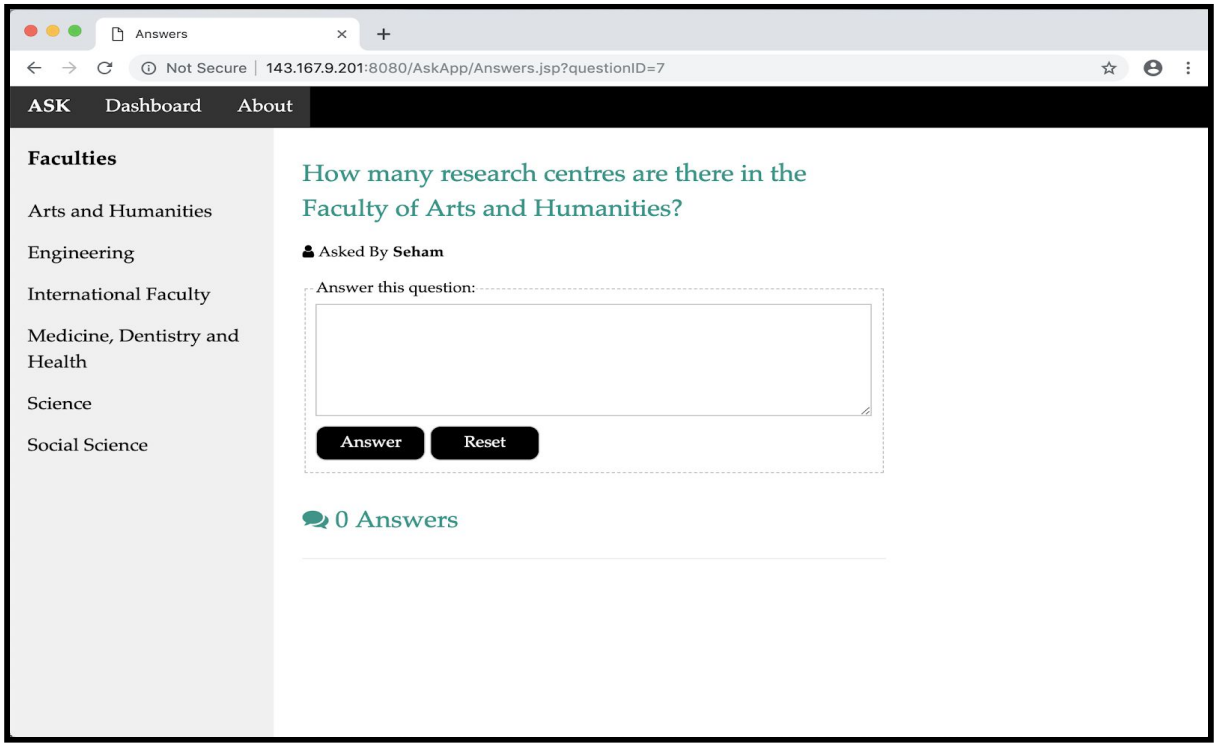


Figure 45:ASK application - a question page with no answers.

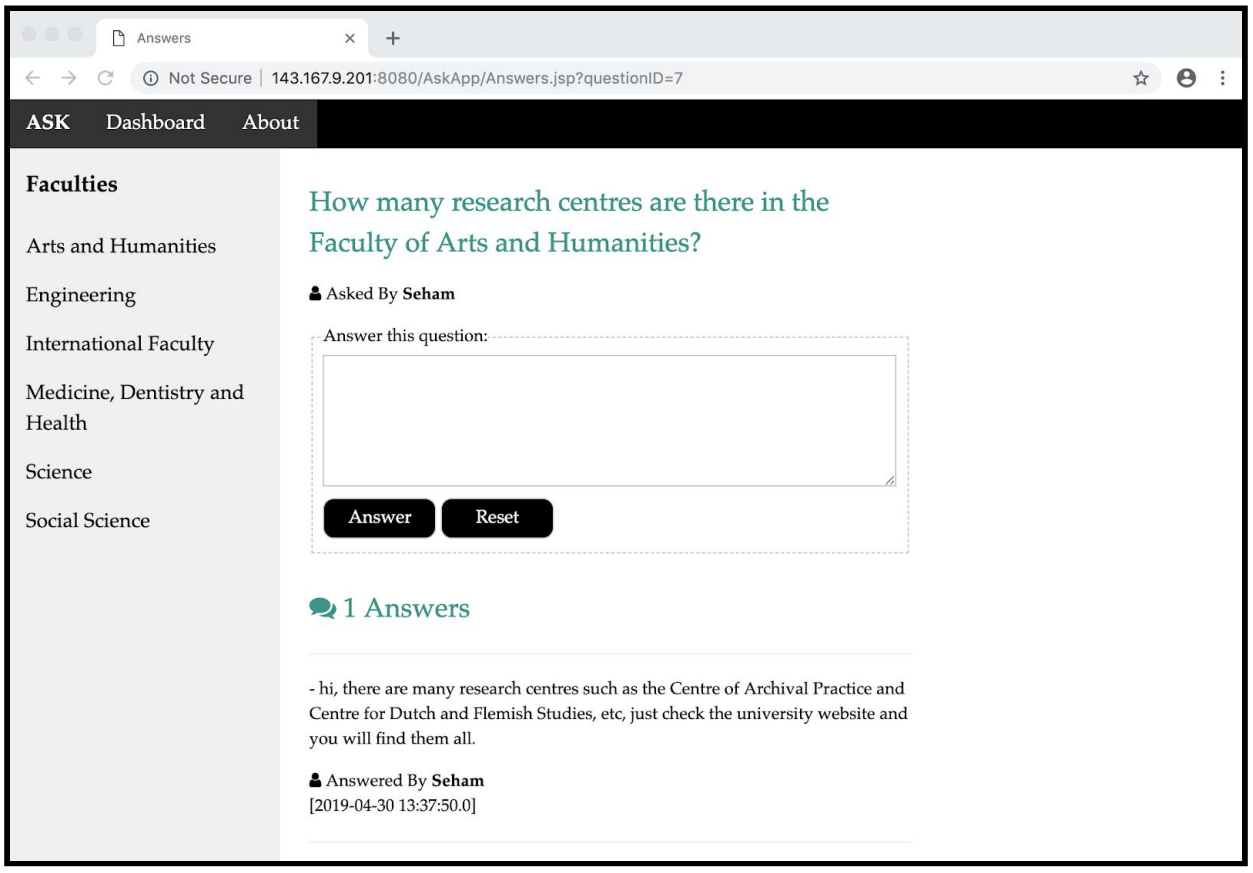


Figure 46:ASK application - a question page after posting one answer.

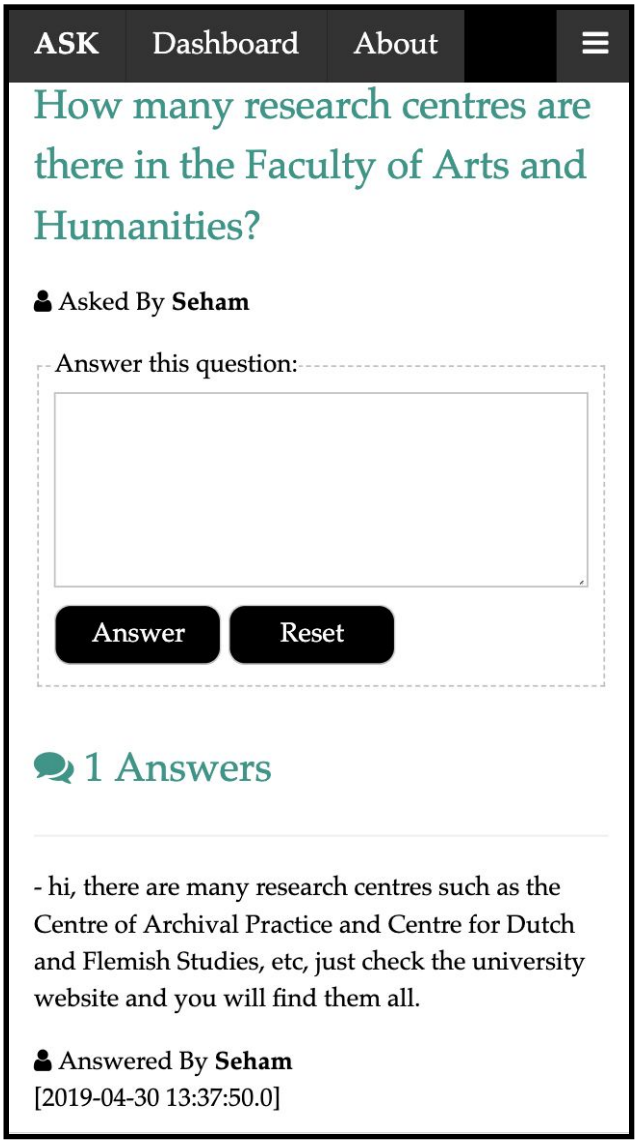


Figure 47:ASK application - question page with an answer (Responsive View).

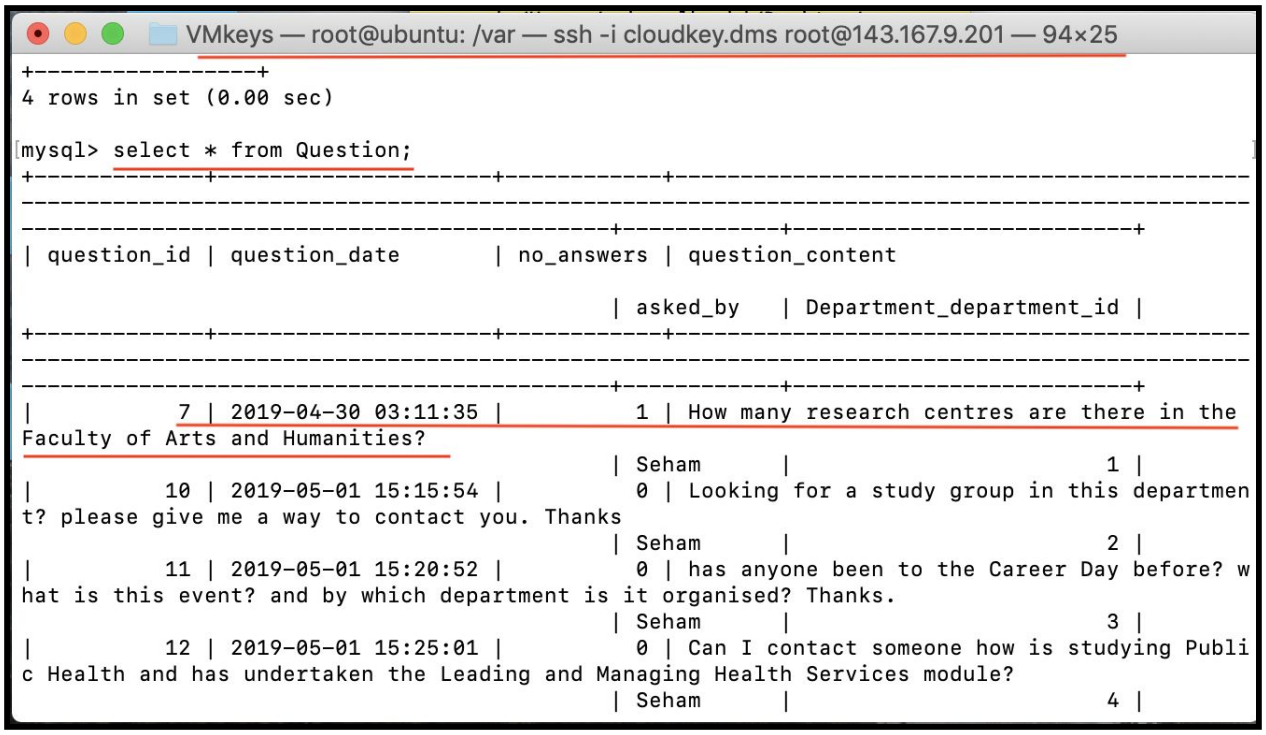


Figure 48:ASK application - the question gets stored in the VM database.

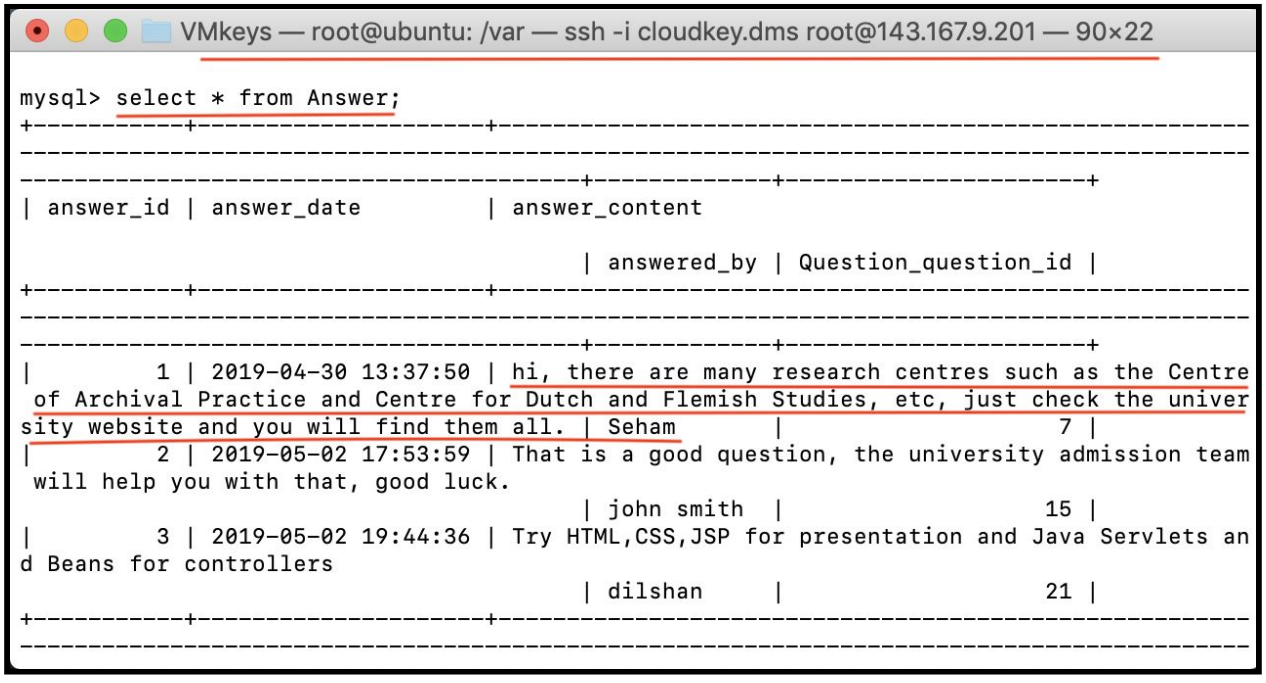


Figure 49:ASK application - the answer gets stored in the VM database.

In conclusion, and as a final step, the Ask application code has been checked and tested by another team member (Dilshan) to make sure that it is readable, understandable, self-explanatory and well documented.

4.5 ISV Collaboration

In order to demonstrate ISV collaboration, an instructions page was added as a guidance to integrate with the platform. These are the same steps followed by the app providers of the platform to connect to the micro services. There are few files which are downloadable and these are required to be placed in the ISV app domain. As it was not possible to find an ISV to collaborate, a test application “MakeANoteApp” was developed and same steps were followed in the instructions page and deployed.

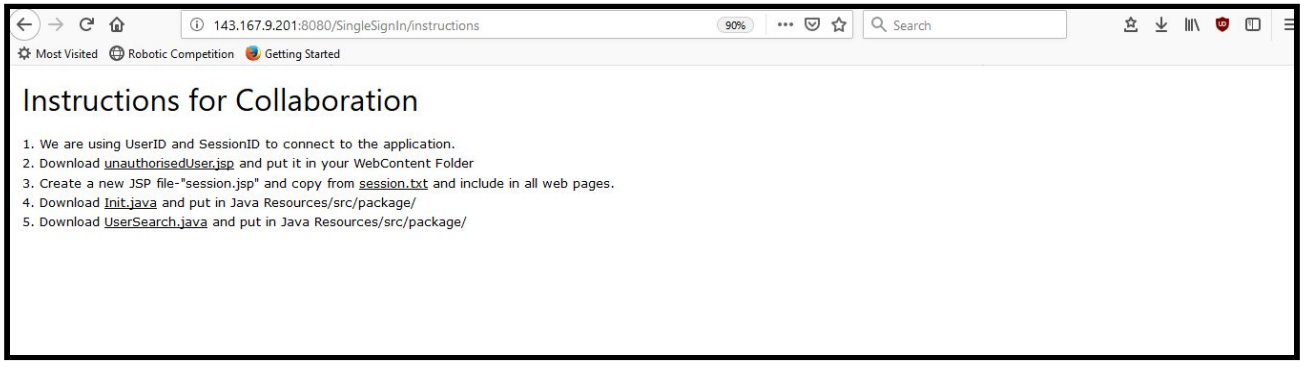


Figure 50:Instructions for Collaboration

5. Security and Loading

There are various standard concerns that should be addressed in order to have our application to work without any loading issues. Firstly, in the process of opening database connections, it is likely for the resources to be highly consumed. Thus, Database connection pooling is used to address this issue as it keeps a cache of open database connections maintained so that the connections can be reused when future requests to the database are required. In this case, when connection is required, pool manager does necessary checks to find out if any unused connection is available and

provides it. It enhances the performance of executing commands on a database. Especially in web-applications our systems get benefited from this behaviour as it can be anticipated that many users(clients) will make multiple requests for a limited period of time and opening and closing database connections will not be feasible in terms of resources.

In Java, DB connection pooling can be implemented with JDBC connection pooling frameworks like Apache Commons DBCP, HikariCP, C3PO.

Secondly, having a pool of servlet instances is an optimal solution that ensures the high performance of concurrent user requests. However, and as it was mentioned in lecture 08 of the cloud computing module, the Tomcat default behaviour to manage the servlets pool is by allocating one servlet instance for each servlet class (Simons, 2019). Another solution is to specify the maximum number of servlet instances, and therefore each request will be served by its servlet instance. However, specifying the maximum number of instances is not an optimal solution, but specifying some numbers of instances could be better.

Cloud data security is vital, and we need to make sure that all the data stored on the cloud is secured. To make our platform secure, we came across the following authentication and authorization methods,

1. HTTP Basic Authentication
2. API keys
3. OAuth 2.0

Let's discuss them one by one,

1. **HTTP Basic Authentication:** In this authentication, the user will just provide the username and password to prove their authentication. This method does not use sessions, cookies, login pages and thus there's no need to handshakes or other complex response systems (Sandoval, 2019).

2. **API keys:** In this approach, the unique value is assigned to the user, signifying the user is known. When the user re-enters the system, the unique key is used as a proof that they are the same user as before (Sandoval, 2019).
3. **OAuth 2.0**: It is for both authentication and authorization. In this approach, when user logs into the system, the system will request for authentication in the form of token. The user will then forward this request to an authentication server, which will either reject or allow this authentication. From here, the token is provided to the user, and then to the requester. Such a token can then be checked at any time independently of the user by the requester for validation and can be used over time with strictly limited scope and age of validity. This approach is more powerful as it allows the user to be authenticated and as well define the validity for the user, so that the user can be deprecated automatically in time (Sandoval, 2019).

In the system, for the users to be logged in, **sessions** should also be taken into consideration. In the logging process session and cookies are required to be stored. HTTP session class can be used for this process and this will facilitate for identifying a user in multiple pages. In this way some features of the system can be made accessible only for logged in users. It will also facilitate to log out inactive users as the session expires after specific time of inactivity. (Simons, 2019) As the system deliverable is a Platform as a service and preview of the apps can be seen without logging in, but when a user attempts to access an app it requires to login. The session data can used to determine if the client is logged in or not and provide appropriate access. Additionally, implementation of sessions will reduce the server-side processing as it only needs to check in the database once to provide required authorisation for the user.

6. Teamwork

Table 1: Tasks at development Stage

Sr. No.	Work	Sanika	Vijeta	Dilshan	Seham
1.	Single Sign In microservice implementation.	✔			
2.	Payment microservice implementation.		✔		
3.	Library Application design and implementation.			✔	
4.	ASK Application design and implementation.				✔
5.	Platform Dashboard- 1. Design and Code Implementation for creating application and peanut account database after registering. 2. Implementation of ISV Application upload functionality. 3. Implementation of running ISV's SQL script to create database on server. 4. Updating the button for new application on Dashboard after uploading new ISV application.	✔	✔		
6.	1. Designing the instructions page for the ISVs to collaborate 2. Development of "MakeANoteApp" to demonstrate the collaboration of ISVs			✔	✔
7.	Report	✔	✔	✔	✔

Table 2: Agreement

Names	Percentage of work	Signature
Sanika Patil	120%	
Vijeta Agrawal	93.3%	
Chathura Dilshan Gulavita Ganithage	93.3%	
Seham Alharbi	93.3%	

Note regarding ISV Collaboration:

As we have not found any ISV for collaboration, we have added “MakeANoteApp” via upload functionality given to ISV.

7. References

1. Docs.spring.io. (2019). 17. Web MVC framework. [online] Available at: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html [Accessed 11 Mar. 2019].
2. Sandoval, K. (2019). 3 Common Methods of API Authentication Explained | Nordic APIs |. [online] Nordic APIs. Available at: https://nordicapis.com/3-common-methods-api-authentication-explained/ [Accessed 11 Mar. 2019].
3. Simons, A (2019). Concurrency, Scaling and Sessions [lecture notes]. The University of Sheffield. Delivered 1 March 2019.
4. Spring.io. (2019). spring.io. [online] Available at: https://spring.io/ [Accessed 11 Mar. 2019].
5. MuleSoft. (2019). Tomcat MySQL Connection - Using JDBC to Connect Tomcat to MySQL. [online] Available at: https://www.mulesoft.com/tcat/tomcat-mysql [Accessed 14 Mar. 2019].
6. En.wikipedia.org. (2019). Apache JServ Protocol. [online] Available at: https://en.wikipedia.org/wiki/Apache_JServ_Protocol [Accessed 14 Mar. 2019].