COM6519-001
180127542
180127667
180219021
180128147

The University Of Sheffield.

Group Assessed Work Coversheet

Assessment Code: COM6519

Description: Stage 1: Product Pitch

Staff Member Responsible: Simons, Dr Tony

Due Date: 15-03-2019 15:00:00

Student Registration Numbers:

180128147

180219021

180127667

180127542

Assessment Code:

COM6519-001

☑ I certify that the attached is all my own work, except where specifically stated and confirm that I have read and understood the University's rules relating to plagiarism.

I understand that the Department reserves the right to run spot checks on all coursework using plagiarism software.

Sheffield Cloudbase Platform

Content Index:

1. Introduction 3

2. Business 3

3. Design 5

3.1 Technologies and its risk analysis 5

3.2 Platform Architecture 7

3.3 App directory structure 10

4. Technical 11

4.1 Platform and Apps 11

4.2 Graphics 13

4.3 Security and Loading 13

5. Teamwork 15

6. References 18

7. Appendix 19

Figure Index:

Figure 1: PaaS Architecture 7

Figure 2: UML Package Diagram 9

Figure 3: App Directory Structure 10

Figure 4: Demo App Homepage 11

Figure 5: Registration Page 12

Figure 6: Successful Registration Message 12

Figure 7: Registered Users Webpage 12

Figure 8: Mockup of the dashboard 19

Figure 9: Single Sign in microservice-Mockup for the User Login 19

Figure 10: Single Sign in microservice-Mockup for User Registration 20

Figure 11: Payment microservice-Checkout Page 20

Figure 12: Payment microservice-Transaction Details 21

Figure 13: Library App-Book Status 21

Figure 14: Library App-User Reviews 22

Figure 15: ASK application-home page 22

Figure 16: ASK application-questions page 23

Figure 17: ER diagram of Library App 23

Figure 18: ER diagram of Ask App 24

Figure 19: ER diagram of both MicroServices 24

Table Index:

Table 1: Tasks at Interim Stage 15

Table 2: Agreement 17

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 check status of the books that they are looking for, as well as other details such as the waiting time and the number of people who are waiting for the same book. Moreover, it will provide features to search for a book in an efficient way and reserve it for a specific interval of time. Additionally, it facilitates the users to view the popularity of the books amongst other readers, view reviews and create 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 point sign-in microservice.
2. **Vijeta Agrawal:** green field development microservice.
3. **Chathura Dilshan Gulavita Ganithage:** Library app
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

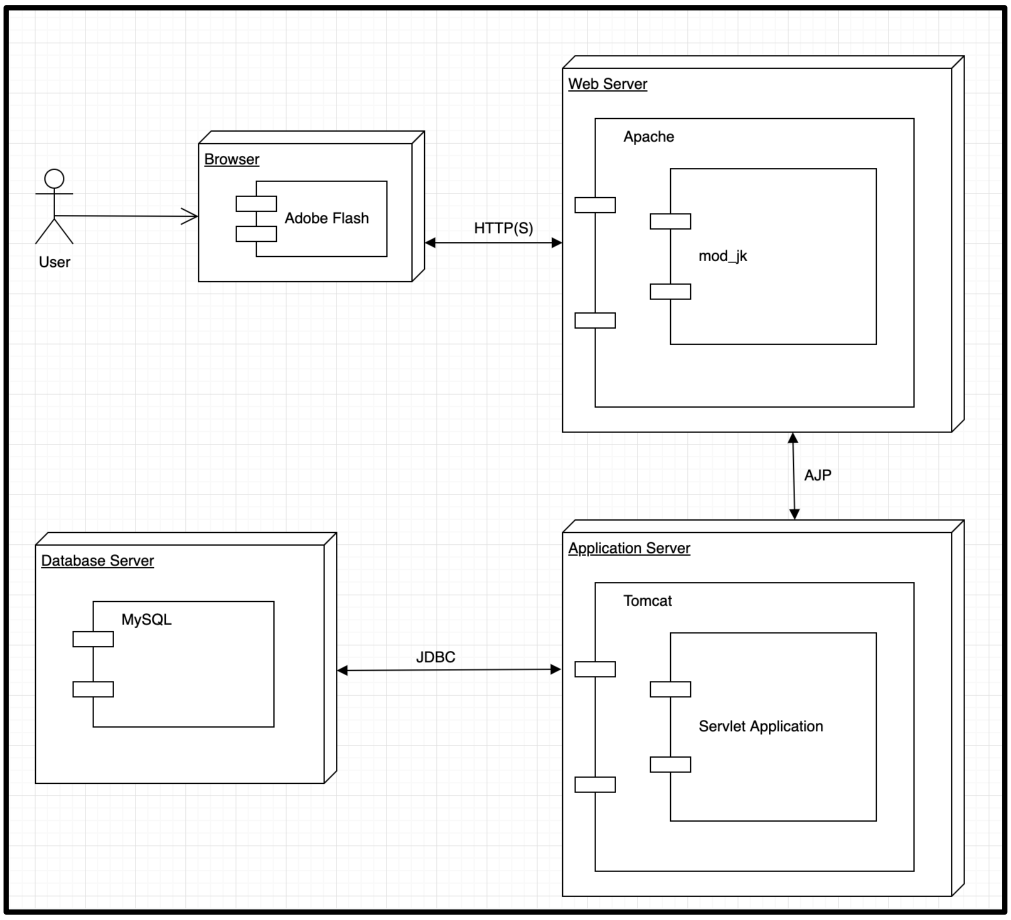


Figure 1: PaaS Architecture

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)).

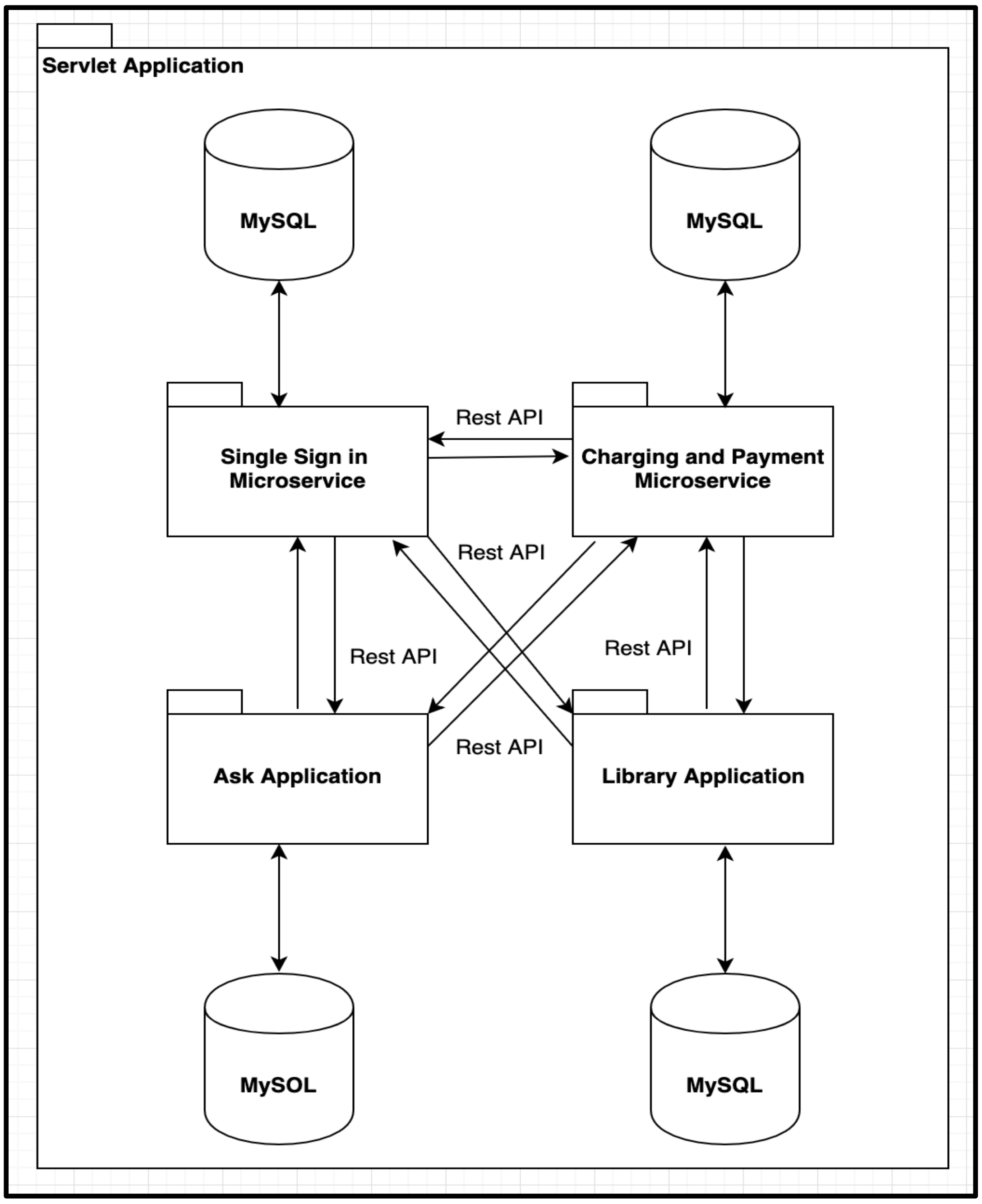


Figure 2: UML Package Diagram

The above figure displays the UML package diagram of Sheffield Cloudbase. It shows how our microservices and applications are connected to each other. Each microservice and application will have its own separate database connected to it. The microservices are communicating with each other using Restful APIs. The applications are also communicating with the microservices using Restful APIs.

3.3 App directory structure

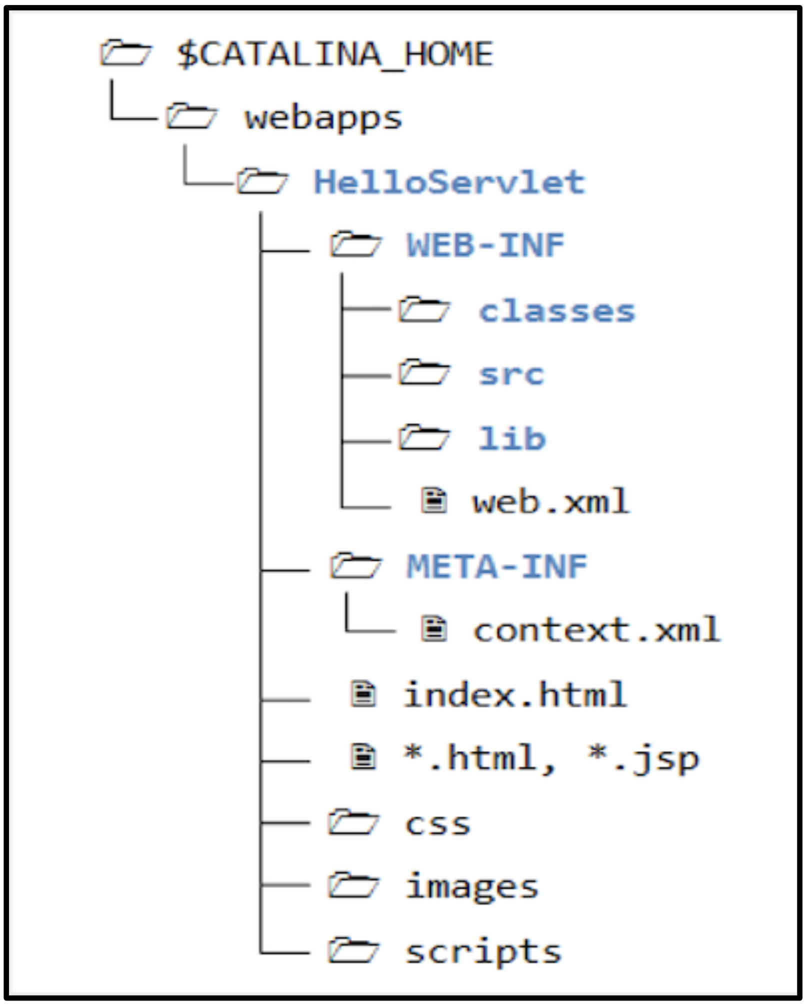


Figure 3: App Directory Structure

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.

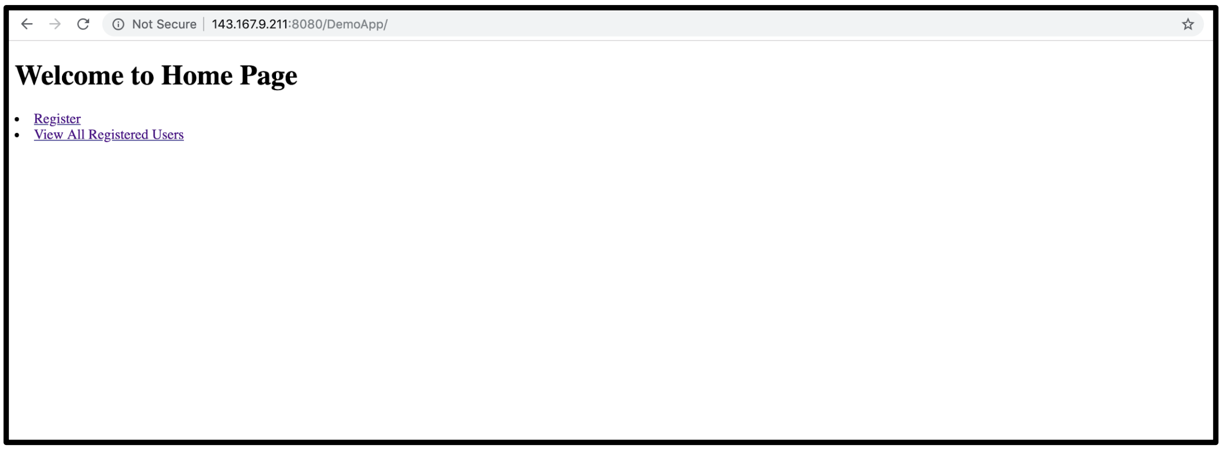
4. Technical

4.1 Platform and Apps

Our DemoApp displays the homepage, and options to navigate to "Register" and "View All Registered Users".

1. On clicking on the "Register" link, the user navigates to "reg.jsp" page. Here the user can enter the username and click on "submit" button. On clicking the "submit", the user gets redirected to "regcheck.jsp" page triggering servlet behaviour and thus the username will be saved in the MySQL database. Here the message is displayed that the user has been successfully registered. Along with this, the user will have the option to "Register" and "View All Registered Users".
2. On clicking on the "View All Registered Users" link, the user will be directed to "populateSqldata.jsp" page. Here all the registered users will be read from the database and displayed on the page. Here the user will also have the option to go back to the "Home" page.

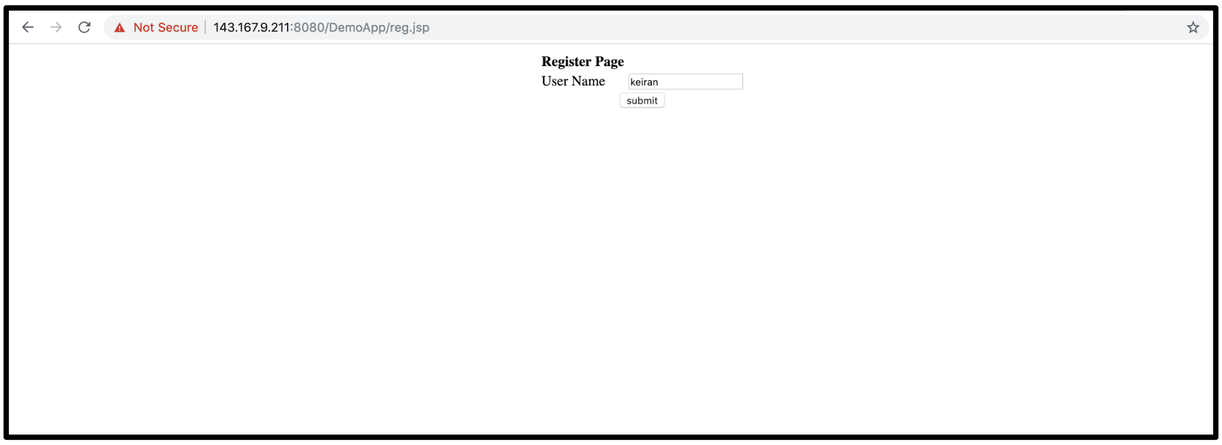
Following are the screenshots for evidence of our working DemoApp:



Welcome to Home Page

- Register
- View All Registered Users

Figure 4: Demo App Homepage

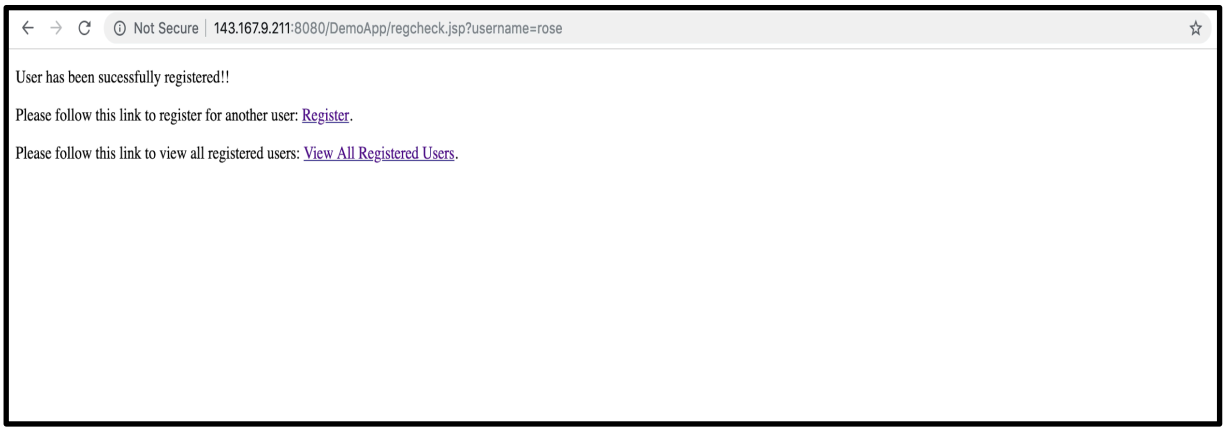


Register Page

User Name keiran

submit

Figure 5: Registration Page

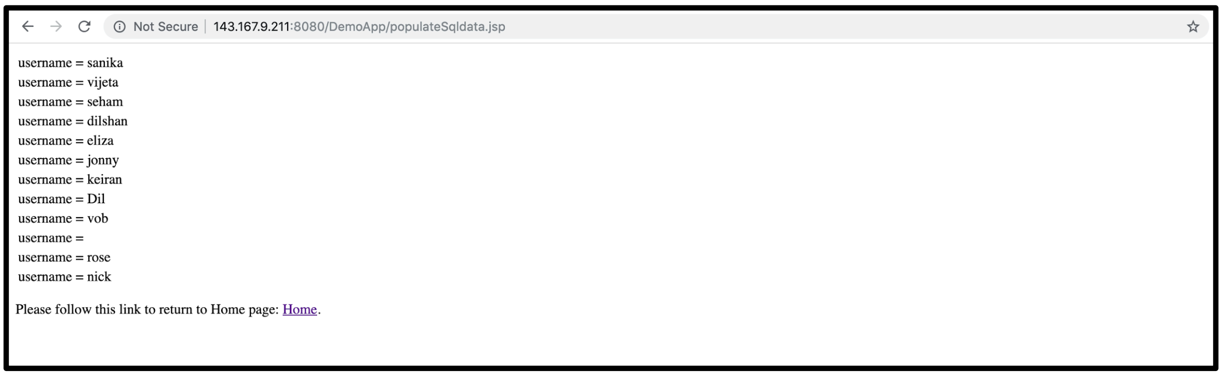


User has been sucessfully registered!!

Please follow this link to register for another user: Register.

Please follow this link to view all registered users: View All Registered Users.

Figure 6: Successful Registration Message



```
username = sanika
username = vijeta
username = seham
username = dilshan
username = eliza
username = jonny
username = keiran
username = Dil
username = vob
username =
username = rose
username = nick
```

Please follow this link to return to Home page: Home.

Figure 7: Registered Users Webpage

4.2 Graphics

The required UI mock-ups of dashboard, microservices and applications are attached in the appendix.

4.3 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).

For our applications, we are planning to implement OAuth 2.0 for authentication and authorization.

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.

5. Teamwork

Table 1: Tasks at Interim Stage

Sr. No.	Work	Sanika	Vijeta	Dilshan	Seham
1.	Setting up the Virtual Machine	✓	✓	✓	✓
2.	Setting up the Tomcat Server	✓	✓		
3.	Research on the technologies	✓	✓	✓	✓
4.	Research on communication between microservices.	✓	✓		
5.	Research on Authentication and Authorization	✓		✓	
6.	Research on how to handle loading issues (scaling, servlet pools, DB connection pools)	✓		✓	✓

7.	Research on web template engines.	✓			
8.	Product pitch advertisement.				✓
9.	Following up with department about setting up the MySQL database on VM.			✓	
10.	Code implementation of DemoApp with SQL functionality, UI development of the webpages, triggering servlet behaviour and deployment of the DemoApp on server.	✓			
11.	Creating Database for the DemoApp on server.	✓			
12.	Creating platform UML deployment diagram		✓		
13.	Creating Mockups for microservices and applications	✓	✓	✓	✓
14.	Creating ERD for databases		✓	✓	✓
15.	Report writing: 1. Introduction 2. Business: Technologies agreed upon and any disagreements.	✓			

	3. Design: Web technologies our team will use to realise this product and risk analysis. 4. Technical: About DemoApp and its evidence of working. 5. Technical: Security - DB connection pool and Authentication & Authorization.			✓	
16.	Report writing: 1. Business: Team product pitch 2. Design: The zipped directory structure and the packing and unpacking processes. 3. Technical: The standard concerns part (Servlet Pools).				✓
17.	Report writing: 1. Design: Platform Architecture 2. Final Formatting 3. Proofreading and editing the content as required.		✓		

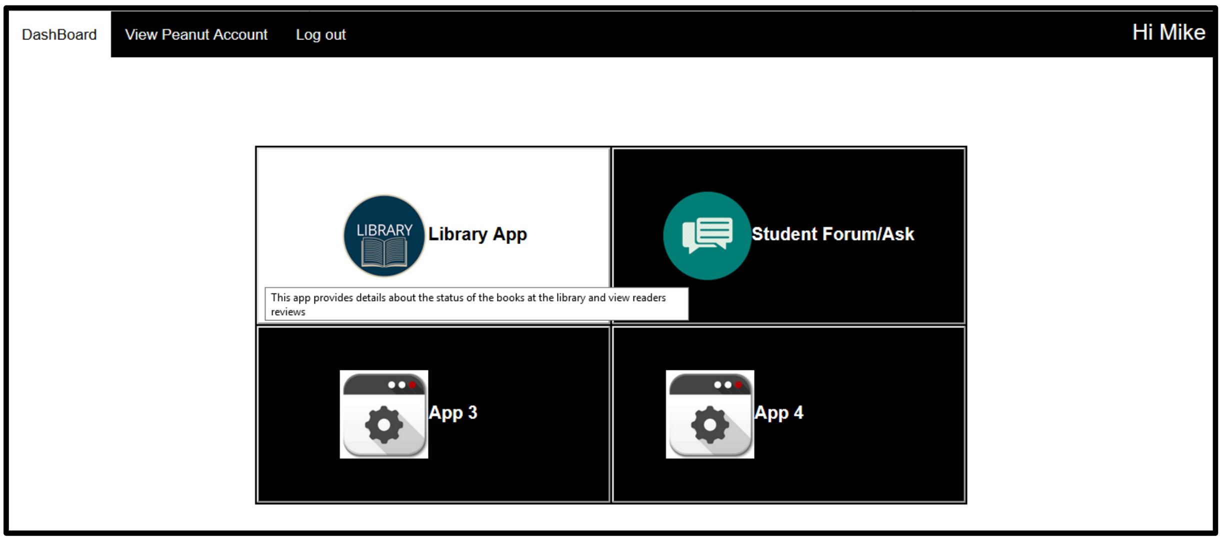
Table 2: Agreement

Names	Signature
Sanika Patil	
Vijeta Agrawal	
Chathura Dilshan Gulavita Ganithage	
Seham Alharbi	

6. 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].

7. Appendix



This app provides details about the status of the books at the library and view readers reviews

Figure 8: Mockup of the dashboard

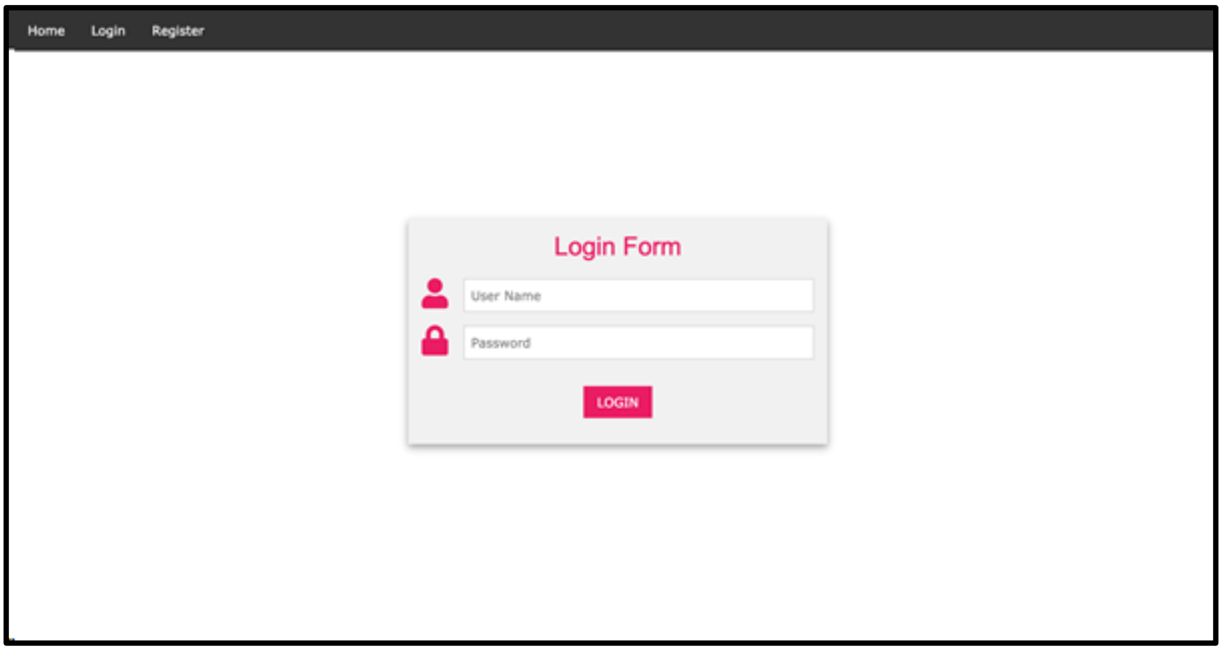


Figure 9: Single Sign in microservice-Mockup for the User Login

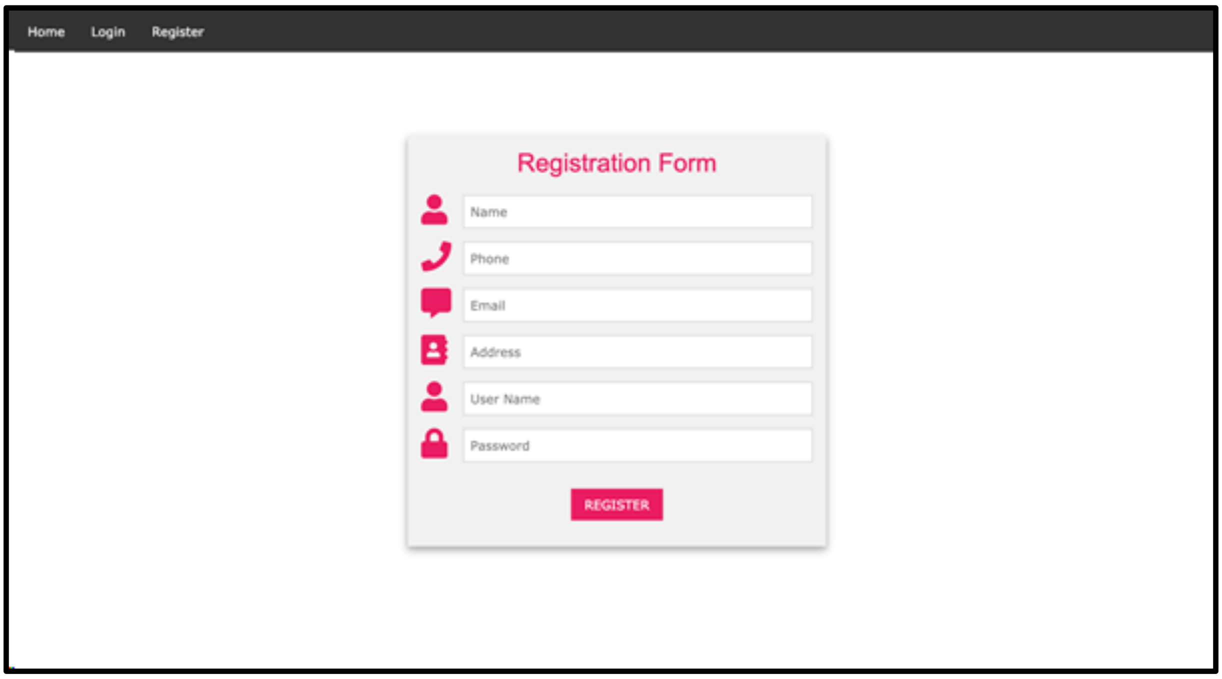


Figure 10: Single Sign in microservice-Mockup for User Registration

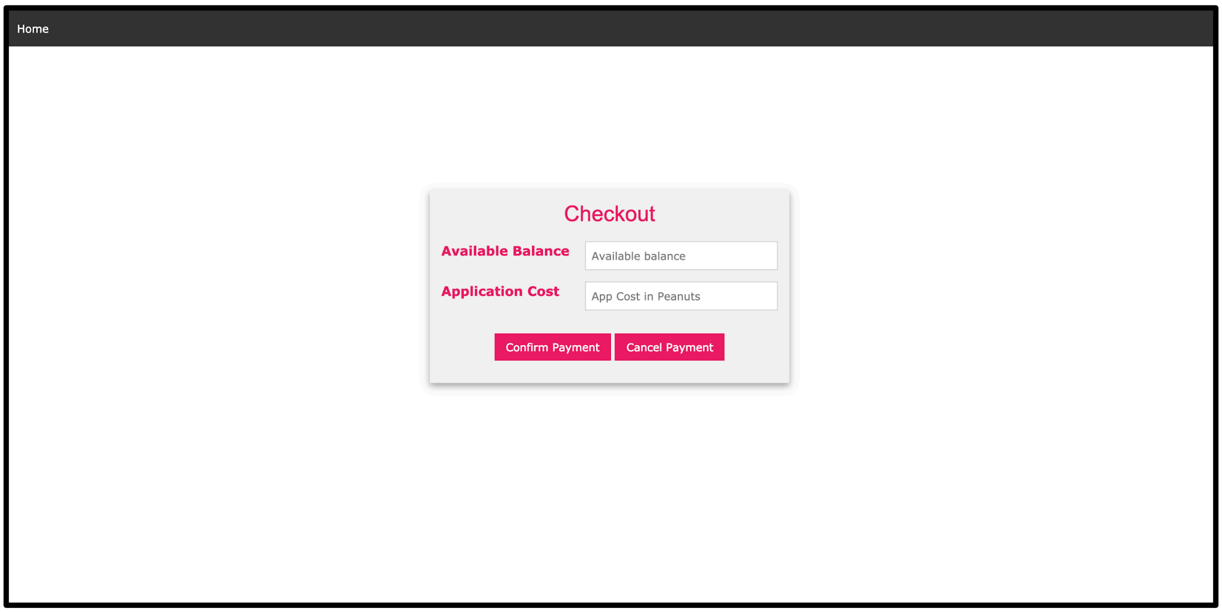
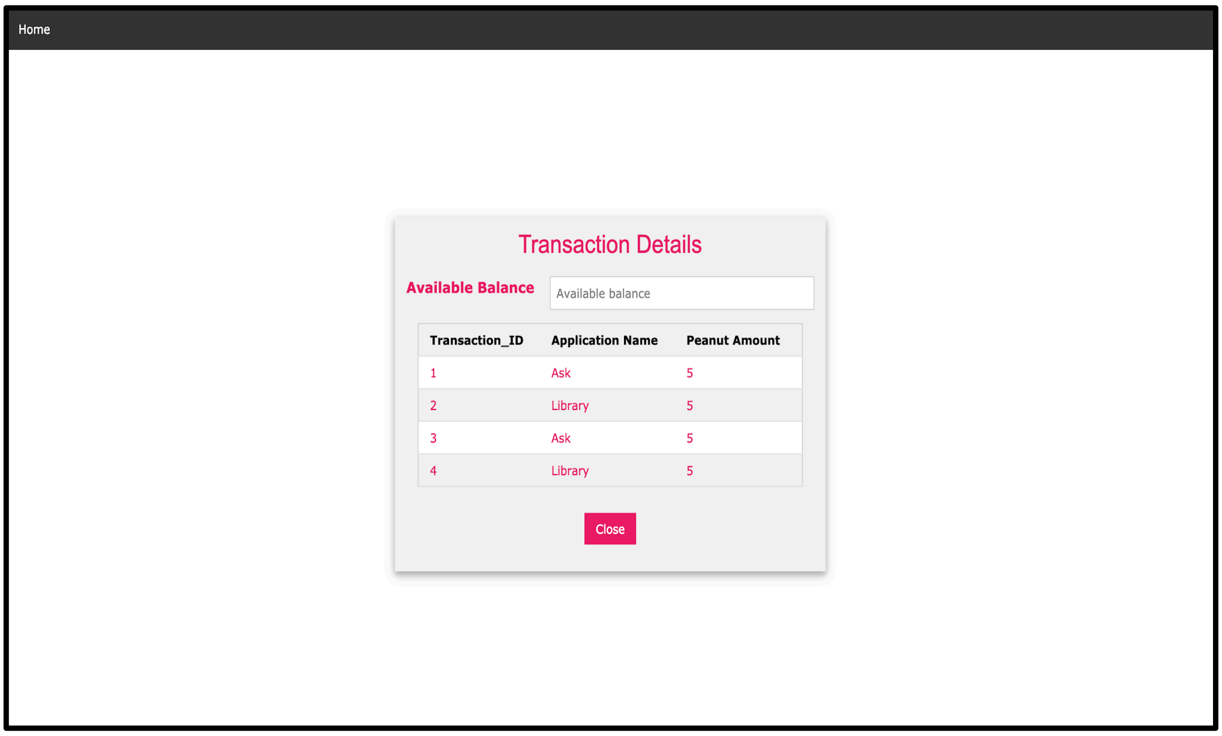


Figure 11: Payment microservice-Checkout Page

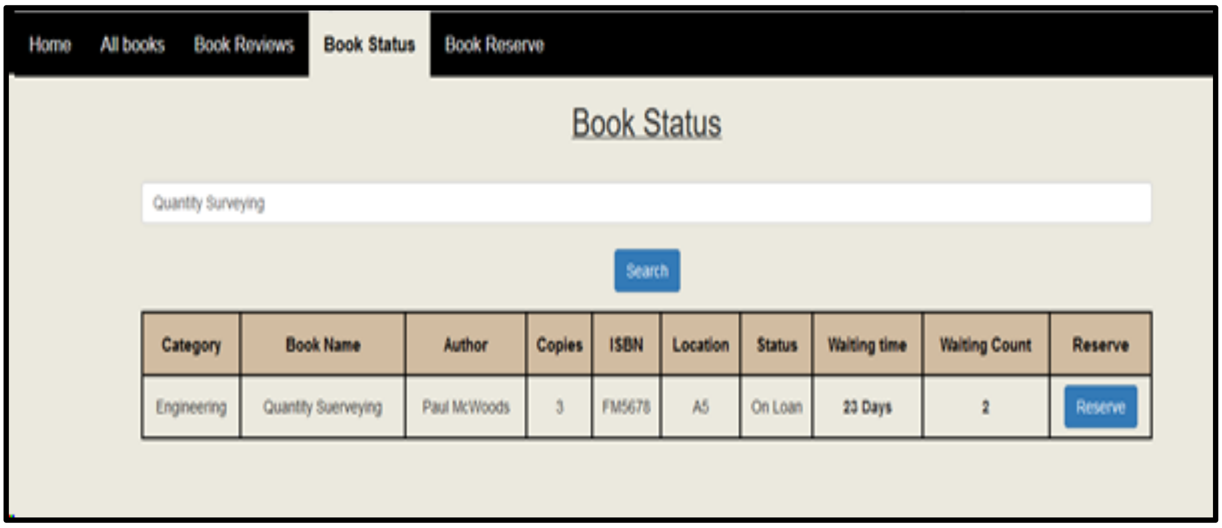


Transaction Details

Available Balance: Available balance

Transaction_ID	Application Name	Peanut Amount
1	Ask	5
2	Library	5
3	Ask	5
4	Library	5

Figure 12: Payment microservice-Transaction Details

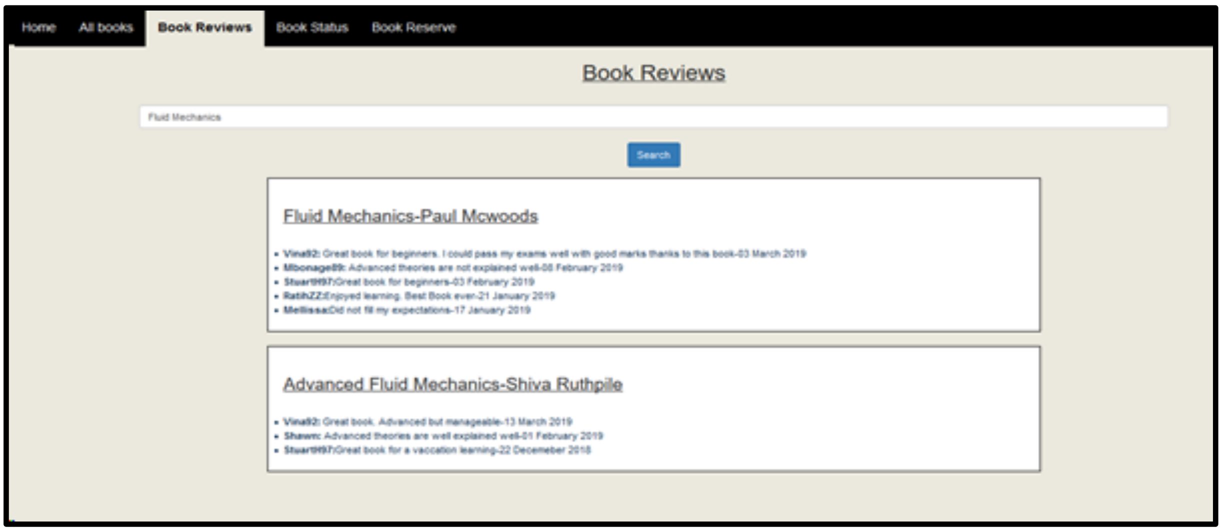


Book Status

Quantity Surveying

Category	Book Name	Author	Copies	ISBN	Location	Status	Waiting time	Waiting Count	Reserve
Engineering	Quantity Suerveying	Paul McWoods	3	FM5678	A5	On Loan	23 Days	2	Reserve

Figure 13: Library App-Book Status



Book Reviews

Fluid Mechanics

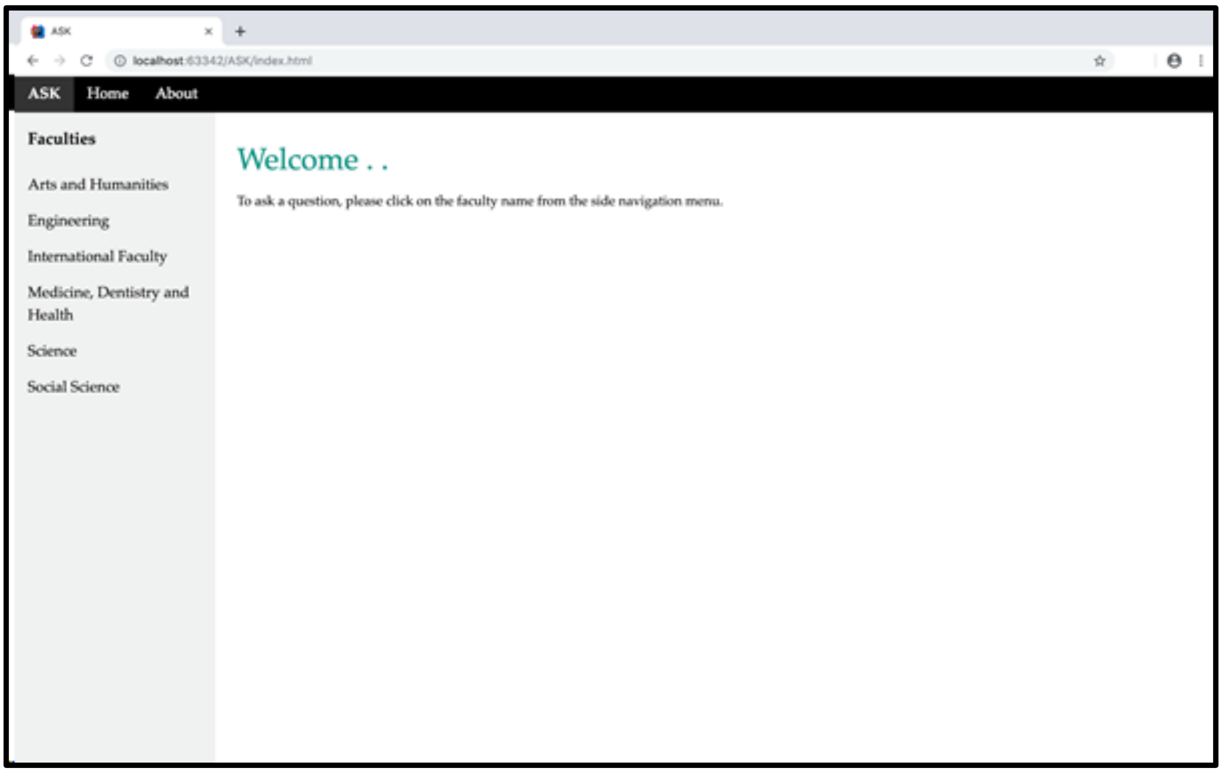
Fluid Mechanics-Paul Mcwoods

- **Vina92:** Great book for beginners. I could pass my exams well with good marks thanks to this book-03 March 2019
- **Mbonage89:** Advanced theories are not explained well-08 February 2019
- **StuartH97:**Great book for beginners-03 February 2019
- **RatihZZ:**Enjoyed learning. Best Book ever-21 January 2019
- **Mellissa:**Did not fill my expectations-17 January 2019

Advanced Fluid Mechanics-Shiva Ruthpile

- **Vina92:** Great book. Advanced but manageable-13 March 2019
- **Shawn:** Advanced theories are well explained well-01 February 2019
- **StuartH97:**Great book for a vaccation learning-22 Decemeber 2018

Figure 14: Library App-User Reviews



Faculties

Arts and Humanities

Engineering

International Faculty

Medicine, Dentistry and Health

Science

Social Science

Welcome . .

To ask a question, please click on the faculty name from the side navigation menu.

Figure 15: ASK application-home page

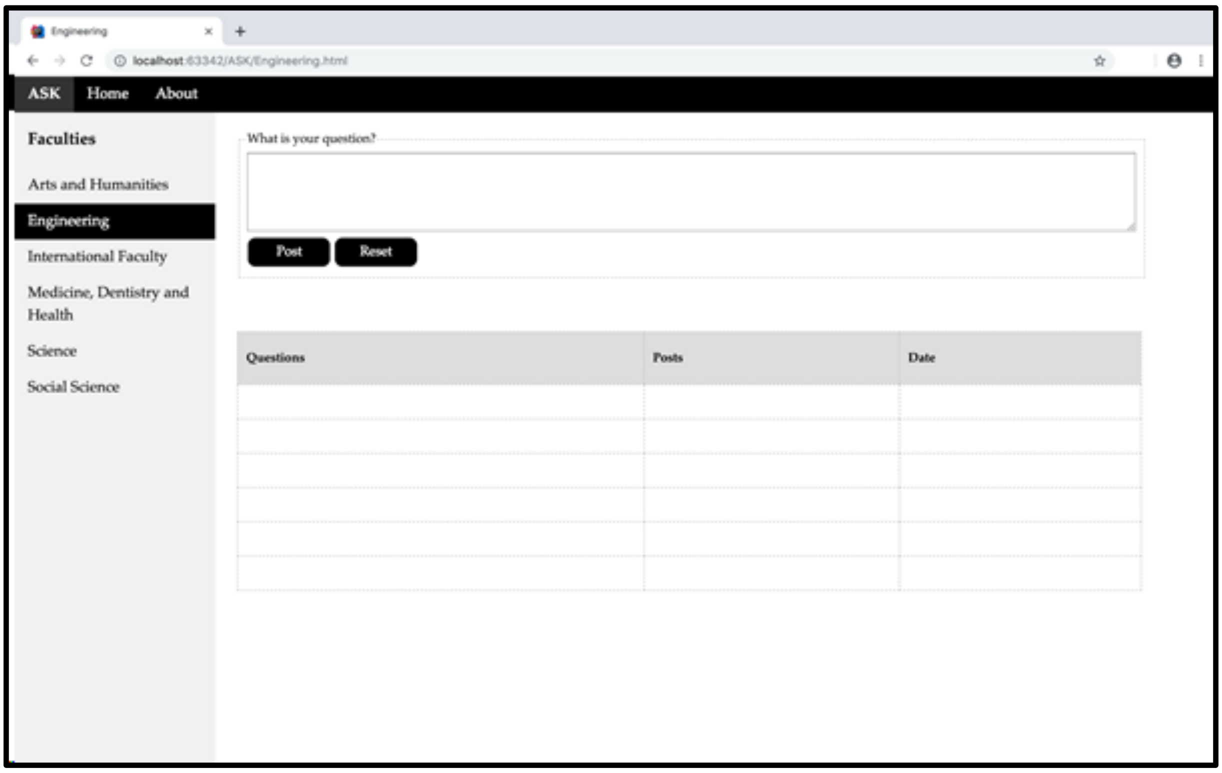


Figure 16: ASK application-questions page

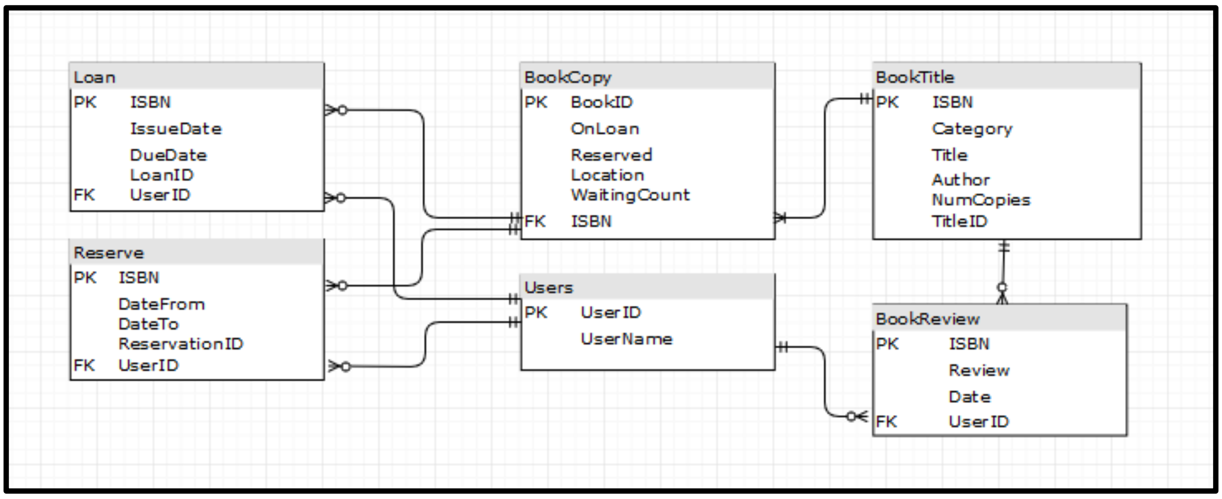


Figure 17: ER diagram of Library App

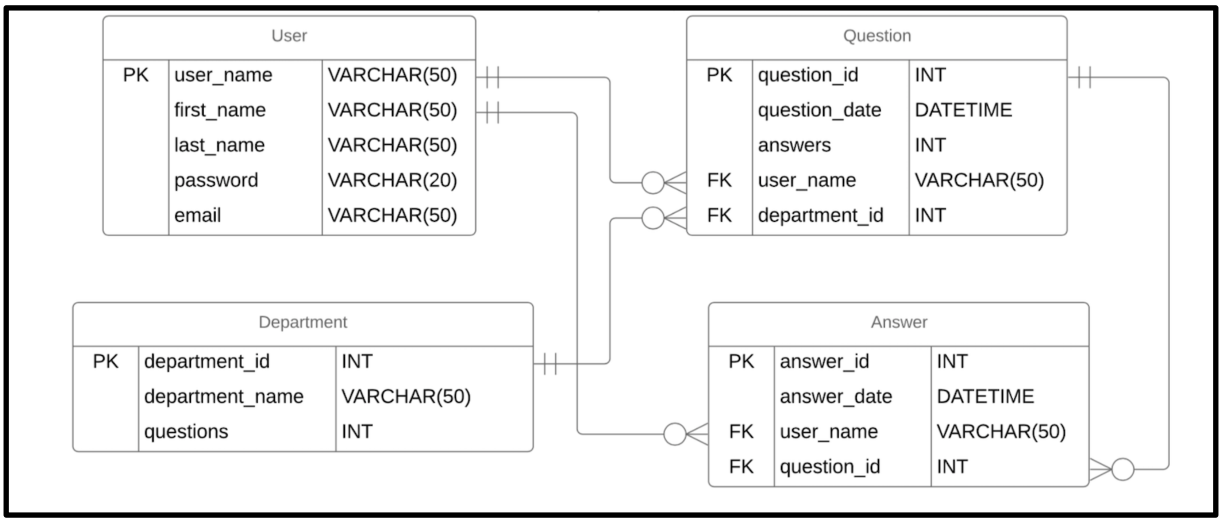


Figure 18: ER diagram of Ask App

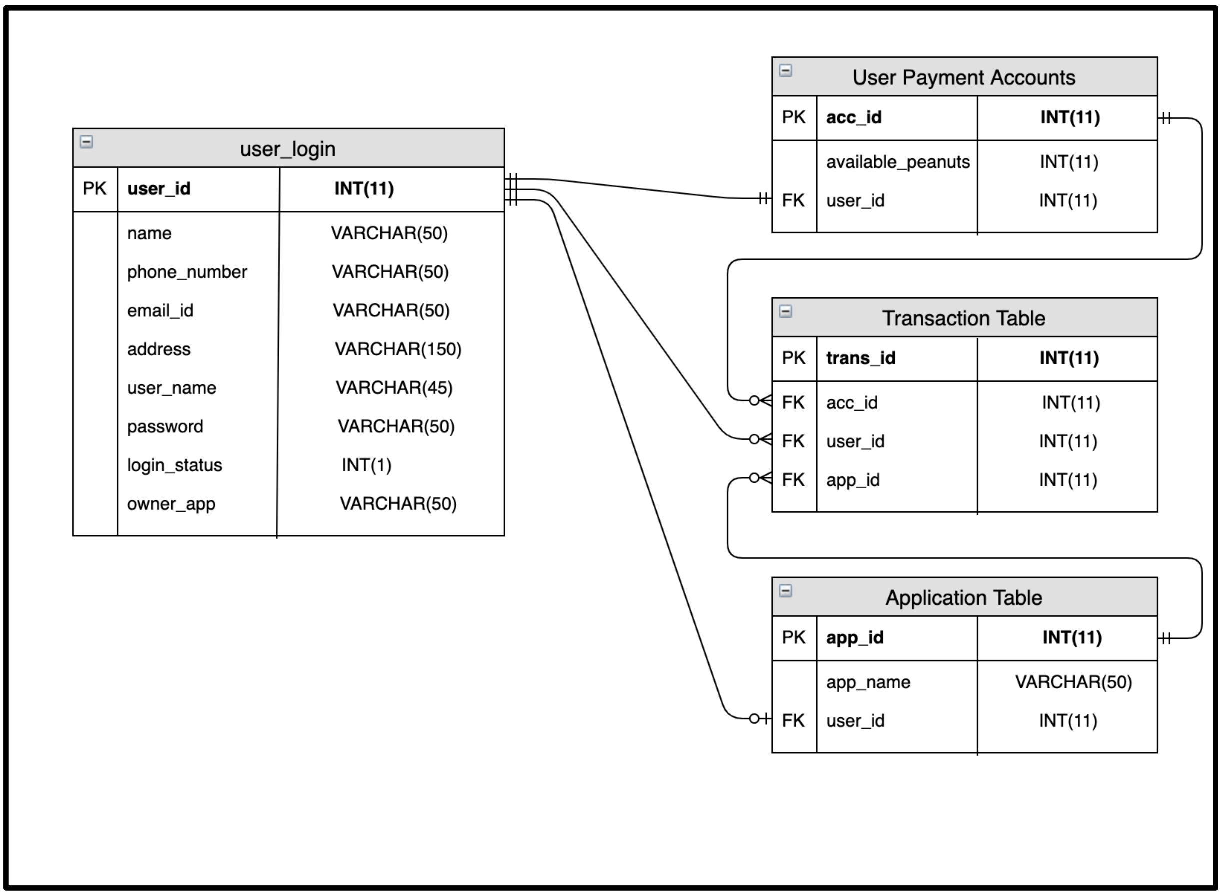


Figure 19: ER diagram of both MicroServices