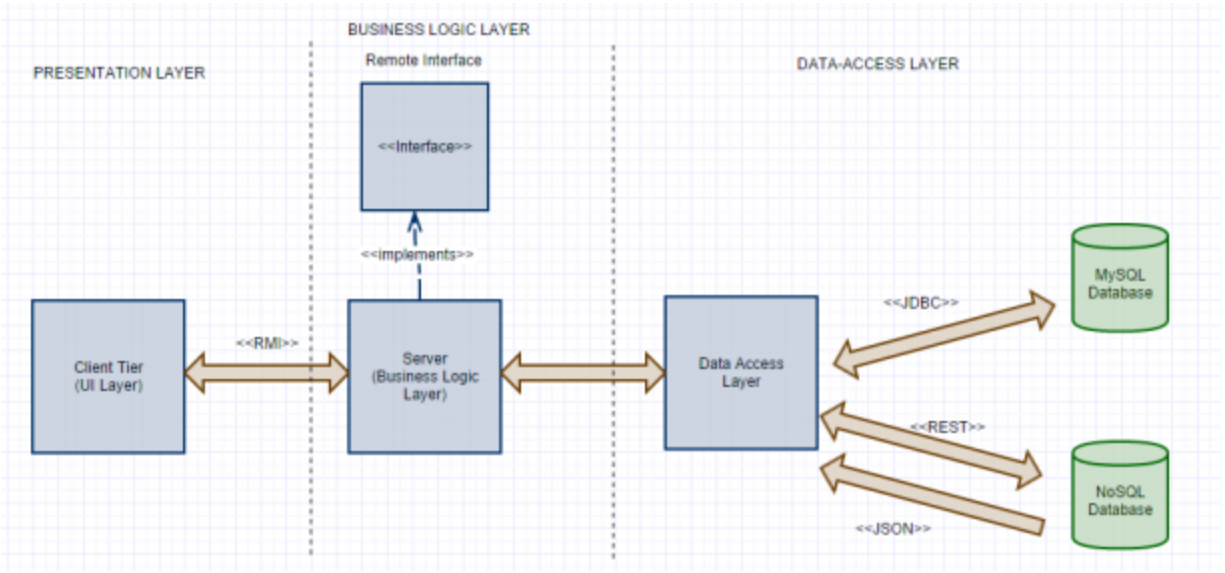
Project Implementation Details:

Initial Architecture



Technologies used in Implementation:

1. Remote Method Invocation (RMI) - This was used as the main technology for implementing the distributed objects application. RMI provides the mechanism by which the server and the client communicate and pass information back and forth.

   RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects.

   Reasons for using RMI:

   - A well-proven technology for developing distributed applications,
   - Java – based technology,
   - Simple Interface Development - We exposed the services provided by Server through a remote interface on which the client invokes the method, and
   - Handles all the low-level networking details automatically.

2. YELP API - We leveraged the Yelp API for fetching the application data which is publically available. We used the capability of the Yelp API [1] version 2.0 by using the Search API to query for businesses by a search term and location, and the Business API to query additional information about the top result from the search query.

   Some of the main features of the YELP API are:

   - Find up to 40 best results for a geographically-oriented search
   - Sort results by the best match for the query, highest ratings, or distance
   - Limit results to those businesses offering a Yelp Deal, and display information about the deal like the title, savings, and purchase URL
   - Identify and display whether a business has been claimed on Yelp.com

   We developed a client (embedded inside our RMI Server Code) which calls the YELP API by passing it the required parameters and gets back the response in the popular JSON format.

   Reasons for using YELP API:

   - Easy learning curve,
   - Humongous database which is essentially the “big data”,
   - Data relevant to our application needs,
   - Easy integration with rest of the application,
   - Data can be easily parsed and inserted into relational database (SQL database),
   - Data can be easily mapped into NoSQL documents,
   - The API uses a standard, secure authorization protocol (OAuth 1.0a, xAuth) for authenticating requests and offers various API methods.

3. JAX-WS (SOAP) – We used SOAP protocol to expose the web services provided by our application to the end-users.

   Reasons to use SOAP:

   - Great library support through JAX-WS,
   - Annotations support,
   - Proven technology.

4. MySQL DB – We chose MySQL as Relational DBMS. MySQL is the #1 database for Web-based applications, used by Facebook, Twitter, LinkedIn, Yahoo!, Amazon Web Services and virtually all the largest Web properties and successful startups.

   Reasons for using MySQL:

   - MySQL is a proven and mature solution,
   - High performance, reliable and easy to use,

5. MongoDB – We chose this NoSQL in conjunction with SQL Database in order to increase the throughput of the system. We have collections containing data about restaurants, hospitals, user authentication details in MongoDB.

   Reasons for using MongoDB:

   - MongoDB is a document database that provides high performance, high availability, and easy scalability.
   - MongoDB stores data in the form of JSON documents which is a direct need of our application,
   - Queries can be much faster than in a relational database where related data is separated into multiple tables and then needs to be joined later.
   - MongoDB works hard to be very easy to install, configure, maintain, and use.

6. Amazon EC2 – We used this cloud hosting service to deploy our application and web server. We exposed our web services on web server as WSDL using SOAP protocol. We deployed the RMI server along with databases on one instance of EC2 and deployed the client (with respect to the RMI server) on another instance of EC2. We built web services over the client whom the end-users eventually use to send a HTTP Request and obtain response in JSON format.

   Reasons for using Amazon EC2:

   - Resolves the hosting of services problem,
   - Can easily acknowledge HTTP request, and
   - Helps in implementing SOA.

7. Mininet- We used mininet in conjunction with JUnit to test out a few basic network configurations to ensure the application could cleanly handle packet loss and high latency. Extreme latency is treated similarly to a disconnect and ensures server threads do not hang when under heavy load from a high number of clients. Mininet also allowed us to configure the project as both an all in one local application (server/DB/client in one host) or as a fully distributed system (server/DB/client all on different machines). NOTE: JUnit testing was done assuming the all-in-one configuration and is always run from the same host as the server, but can be modified to test remote clients and databases.

   Reasons for using Mininet
   - Lightweight and portable VM for testing
   - Easily modifiable test environment
   - Cheap and effective way to test out network loads without using additional hardware
   - Free and well documented

8. Apache JMeter – In our application, we have followed a service-oriented architecture. We want to test and check the performance of our web services for multiple users. Apache JMeter is primarily used for load testing of our application.

9. JUnit- As above we used JUnit to test the functionality of the RMI server when it is run as a standalone on either a local machine (via Eclipse) or deployed on mininet. The unit tests simulate a client in that they use the interface to test the functions as opposed to directly testing the server class itself.The unit tests themselves test the connection methods in regard to user validation using dummy data from our test data scripts, and then test the functionality of the server by testing each of the ‘get’ data methods.