RESTful Web Service Modeling with NoSQL Database

Jiajie Wu
Yue Hu
Zhijun Jiang
Table of Content

1. Preface ........................................................................................................................................3
2. Acknowledgements ....................................................................................................................4
3. Abstract ........................................................................................................................................5
4. Introduction ..................................................................................................................................6
5. Theoretical Bases and Literature Review ..................................................................................7
   5.1 SOAP based Architecture .......................................................................................................7
   5.2 REST: Resource-oriented Architecture ..................................................................................8
   5.3 RESTful Web Service with Node.js and MongoDB ...............................................................9
6. Hypothesis ....................................................................................................................................9
7. Methodology ................................................................................................................................10
8. Implementation (refer to source code in appendix) .................................................................10
9. Design Document .......................................................................................................................11
   9.1 Introduction ..........................................................................................................................11
   9.2 System Overview ..................................................................................................................11
   9.3 System Description ...............................................................................................................12
   9.4 Conclusion ............................................................................................................................16
10. Data Analysis and Discussion .................................................................................................16
11. Conclusion and Recommendation ...........................................................................................21
12. Bibliography ............................................................................................................................22
13. Appendix ....................................................................................................................................23
Preface
The Representational State Transfer (REST) style Web Service is lightweight and becomes popular: major Internet companies such as Google, Amazon, and Yahoo have all published REST APIs. We propose an approach using Node.js with MongoDB as database. Therefore, we will implement the RESTful Web Service using Node.js and generate different workload to measure performance of the system. We illustrate how we model a Web Service with a Recipe Website. The conceptual model of comment operation will be showed in this paper.
Acknowledgements:

The authors would like to thank Irum Rauf, Anna Ruokonen, Tarja Systa, Ivan Porres, and Dept. of Software Systems Tampere University of Technology, Tampere, Finland and Dept. of Information Technologies Abo Akademi University, Turku, Finland. Finally and most importantly, we would like to thank Professor Wang for his explanation and instructions during the project.

Abstract:
Representational state transfer (REST) architecture is now widely used. RESTful architecture enables web service exhibit its functionality in the exposed resources. To build a Web Service with extensible scalability, we investigate some architecture styles and modeling method. This paper uses RESTful Web Service to build a conceptual and behavior model of a recipe website and measure the performance of Web Service components.
I. Introduction:

With the emergence of cloud service, nowadays a web-based organization need to store and access data frequently in a distributed system. In production environment, it is essential to integrate data with the existence of heterogeneity in platforms, programming languages and data structures. In order to efficiently organize and utilize the distributed system, it is important to design a Web Service architecture adaptive to the cloud environment.

Web services can be developed in a Remote Procedure Call (RPC) manner or as a Representational State Transfer (REST) style. The RPC-styled web service is operation centric. It exposes its functionality in the operations advertised on its interface. The main mechanism behind RPC is message passing: a client sends a request message to a server with parameters for a certain procedure. The server execute the procedure and sends back response to the client. Simple Object Access Protocol (SOAP) is a successor of RPC and has been a well-accepted architecture style. Despite its extensibility and independency, verbosity of SOAP resulting from envelope wrapping becomes a major obstacle for performance when data exchange becomes more and more frequent in a distributed system.

On the other hand, the Representational State Transfer (REST) style Web Service is lightweight and becomes popular: major Internet companies such as Google, Amazon, and Yahoo have all published REST APIs. REST web services follow a different architectural style and thus require different design philosophy and techniques. The REST style architecture is resource oriented and exhibits its functionality in the exposed resources. A RESTful web service is designed such that its interface offers addressability, connectivity, uniform interface and statelessness. The advantages of RESTful systems is that they are highly scalable and highly flexible. Because the resources are accessed and manipulated using the four HTTP verbs, the resources are exposed using a URIs, and the resources are represented using standard grammars, clients are not as affected by changes to the servers. Furthermore, RESTful systems can take full advantage of the scalability features of HTTP such as caching and proxies.

In order to specify how resources and methods of composite REST web service are mapped to resources, it is important to model a composite RESTful web service for its static and dynamic behavior using class diagram, activity diagram and state machine diagrams. In this model, the addressability feature of REST requires that any relevant information related to the service is exposed as a resource. Each resource has one or more unique addresses and has one or more representations that are accessible remotely. In order to achieve connectivity resource representation should contain links to other resources such that the graph formed by resources and their links is connected. For a uniform interface, all resources are manipulated using the same set of methods. In the case of HTTP web services the methods are GET, POST, PUT and DELETE.
The paper is organized as follows. Section 3 gives an overview of Web Service under the cloud environment and detailed modeling using REST style architecture. Section 4 specifies why Node.js framework and MongoDB are suitable to implement a RESTful Web Service. Section 5 provides details of how to measure performance of the Web Service.

III. Theoretical Bases and Literature Review

A Web Service defines the communication rules over a network through programmatic interface using standard protocols. A well-designed Web Service ensures that service requester and provider can exchange data efficiently over the internet. Since the two entities are often in two different software systems, heterogeneity and interoperability become important issues. In this context, Service-oriented Architecture (SOA) is a feasible solution because it provides abstraction from underlying complexity and independency from implementation technologies. The Simple Object Access Protocol (SOAP) based framework with XML and WSDL standards has been a popular SOA architecture. This framework has the advantage of loose coupling, seamless interoperability, and good scalability.

3.1 SOAP based Architecture

![Figure 3.1 Communication Process in Web Service](source)

SOAP was designed as an object-access protocol in 1998 by Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein for Microsoft, and is now maintained by the XML Protocol Working Group of the World Wide Web Consortium. Figure 3.1 shows a set of Web Services, based on Web Services Description Language (WSDL), Universal Description Discovery and Integration (UDDI), and SOAP. As shown in the graph, the server provide SOAP based services that are published on the registry UDDI. The client shall specify request with criterion and search for a corresponding service on the UDDI.
registry. The client uses binding operation to connect to the server and request for remote service using SOAP APIs.

Since SOAP is a messaging protocol with a set of constructing rules, it is not efficient to serialize and deserialize information into SOAP message. Besides, since clients cannot obtain useful information directly from URI, it is also impossible to take advantage of proxy and cache server. In sum, redundant information and complexity of this framework result in a less satisfying performance when there is more and more frequent information exchange nowadays.

3.2 REST: Resource-oriented Architecture

Documented by Roy Fielding, Representational State Transfer (REST) is a Resource-oriented Architecture (ROA). Resource can refer to any mapping to a set of entities in a service interface. Specifically, a resource need to be referable with an address. The address is a Uniform Resource Identifier (URI) to location the resource. Representations are any valuable information about the states of a resource, in the form of bytes stream and metadata.

A REST architecture is defined by four attributes:

- **Addressability**: As mentioned above, any resource can be retrieved using URI so that the architecture no longer needs an extra resource locating mechanism such as UDDI.
- **Connectedness**: Representation contains hyperlinks to other resources so that state transfer can be implemented.
- **Statelessness**: Server does not remember the state of applications so that each request shall contain all necessary information to complete the service.
- **Uniform Interface**: Resources are manipulated using the standard HTTP methods.

A RESTful Web Service following the four attributes is well adaptive to the distributed trend nowadays. Addressability contributes to easy accessibility, which is important under frequent data exchange in a cloud environment. The statelessness feature contributes to higher reliability in that independency of each request so that failure of one request has less potential influence on others. Since the no application state needs to be recorded by the server, the server can simultaneously server more request, which can improve the scalability of the whole system.

3.3 RESTful Web Service with Node.js and MongoDB

While the previous research implement RESTful Web Service with Django framework and Ruby on Rails, we propose a different implementation approach using Node.js with MongoDB as database. The detailed
reasons why Node.js platform and MongoDB are better choices in a cloud environment are stated as follows.

Node.js is a cross-platform runtime environment. It was invented with event-drive model adapted to the Web. The platform uses a non-blocking I/O model and single-thread event-based loop. Under this mechanism, the thread will not be blocked while other operation is processing so that Node.js can keep many collections alive while still severing incoming connections. This feature especially useful to accommodate real-time Web application. Node.js is implemented with JavaScript because it has no uniform I/O API so that it could be designed with the non-blocking model. Node.js’ event loop does not need to be called explicitly, instead any I/O related operation must use a callback so that the server could proceed to deal with callbacks. While millions of connections can be handled simultaneously, Node.js provides a highly scalable platform.

MongoDB is document-oriented NoSQL database designed for ease of development and scaling. While traditional database has rigid schema, MongoDB uses a BSON document. A BSON is a JSON-style document that takes all data stored in a row that spans multiple tables of relational database and aggregates it into a single document. With this flexible data model, it is easier to distribute resulting document and improve performance. Furthermore, MongoDB has an auto-sharding mechanism to redistribute data and handle load balancing. Data migration is flexible so that bandwidth is required at a minimum level and the whole system has higher scalability. Besides, MongoDB’s ability to store JavaScript objects natively saves time and processing power. Instead of a domain-specific language like SQL, MongoDB utilizes a simple JavaScript interface for querying. Looking up a document is as simple as passing a JavaScript object that partially describes the search target. In conclusion, MongoDB is not only suitable for distributed system, but also well adaptive to the Node.js based RESTful Web Service.

**IV. Hypothesis**

The goal of our work is to implement a RESTful Web Services using Node.js and MongoDB. Node.js combined with a document database and JSON offers a uniform JavaScript development stack. We suppose this lightweight framework cloud comply with the features of a RESTful Web Service and would lead to a satisfying performance. Our hypothesis is that the response time of service shall increase linearly with the increase of workload intensity. The increasing speed of response time is a measurement for Web Service scalability so that slower increasing speed indicates a higher performance in system scalability.
V. Methodology

In this section, we illustrate how we model a Web Service with a Recipe Website. The conceptual model is shown in Figure 4.1. “Search” is a POST operation while “Details” is a GET operation. The “Comment” is a collection resource that has four resources: “getComment()”, “addComment”, editComment, and “deleteComment”.

We use MongoDB as our database. The input data includes recipe details. Input data are manually collected and input into MongoDB. According to the functionality of our front end project, we will have 3 tables which are “Search_entries”, “Details”, and “Comments”. They are related, details table has a foreign key points to IDs of entries in “Search_entires”. The “Comments” table has a foreign key points to IDs in “Details” table. When web service receive a HTTP call, it will retrieve data from MongoDB, and then send back to front end in JSON format.

We have implemented the RESTful Web Service using Node.js and generate different workload to measure performance of the system. Our major measurement calibrator is response time. If the average response time does not increase significantly with increasing workload, we can expect the system to have a satisfying scalability.
We use Apachebench to generate request and measure performance of our web service. Apachebench is a single-threaded command line computer program for measuring the performance of HTTP web servers. We can set number of requests, concurrency level, request type (get, post, put, delete). Since post is a most representative request with more user interaction, we have tested the performance using one of the post type API: /search. The input is a JSON file including search word. We can set Apachebench with -p as POST type request and -T “application/json” to define the input type. A sample command of Apachebench is “ab -n 1000 -c 200 -p data.json -T 'application/json' http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/search”.

VI. Implementation

Source Code – Please refer to source code in appendix.

Design document

6.1. Introduction

This software design document describes the architecture and system design of building a RESTful web service modeling with Node.js and MongoDB. Our goal is to try to study a new cloud web service computing technique and build a software system based on what we have learned.

6.2. System Overview

This software system our built is website provide a service about food recipes sharing and commenting. For the front-end side, we use a new language called Ember.js which a new framework for creating web applications; for the back-end side, we use node.js which is a platform for building fast, scalable network applications; for the database, we use MongoDB which is an open-source document database with Agile and Scalable feature. Therefore, the software system we built is combined three new cloud web technologies which make our system faster and scalable.
6.3. System Description

Our system starts with a login screen which requires user to input username and password. After user login, the system will process the input information and validate it, if it is correct then process next step; if not, system will show a username or password error message and stay in the login screen. After login success, system will route to a homepage called “iFood Home”. In the home page screen, some dish images is showed in a slide windows and the system will provide three functions: create a new recipe by user, input a keyword to search existed recipes in system and display recipe category items in screen in order to let user do the quick search. Following is the three functions detail:
The Create New recipe function: when user clicks the “+new” button in the home page, system will route user to an “Add New Recipe” page. In the page, user is required to input the details of a recipe: name of the recipe, image URL which for showing a dish, choose a category type which can add more,
Ingredients and cooking method which can add more steps. After inputting, user click the save button, then system will send those information to backend and create a new record and save it to database.

The Search keyword function: when user input a keyword on the search field in the home screen, then system will get the keyword and perform a query search in database; if it return any results, then system will display a result screen which shows match dishes with basic information, if user click on of them, system will show a dish detail screen contains all the detail information of the dish. The category quick search function: there are some categories showed in the home page screen, user can click on of them, then system will perform a database query search based on the category keyword, and then display a result screen just like the search keyword function.
In the recipe detail screen, there is a comment part which can let user see all existed comments about the recipe, let user add a new comment and delete the comment which created by the current user. For the comment part, system provides three functions: add a new comment, show comments and delete a comment. The follows will show how they work:

**Add a new comment function:** there is a comment part in the bottom of detail page, when user click the “Have comment?” field, system will shows a field to let user ranking which is up to five stars and adding a comment for the recipe. After user click “Submit Comment” button, system will send to backend and create a new comment record to save it in database and system will refresh the comment screen to show the all comments including the one user just created. **Show comments function:** when user click the “Comments” field, system will perform a database query search, the comment partial screen will show all the comments of the recipe if there is any results return. **Delete comment function:** when user click the “Comments” field, user will see all comments including the one which user created, and there is a delete button showed to let user to delete the comment. After user click the delete button, system will perform a database delete record query to remove the comment and system will refresh the comment screen to show the all comments except the one user deleted.

### 6.4. Conclusion

Before start programming, we have considered other RESTful programming language technique, such Ruby on Rails or Django. Because our goal is to build a faster and scalable cloud web service computing, so we choose a new technique Node.js and MongoDB in the backed side, and also think about the programming language homogeneity in order to perfume a fast way coding, we choose Ember.js as the frontend side language. We will also do some benchmark tests to show how those three new techniques combine together beat the old web service technique like SOAP.

**Flow Chart** – Please refer to program flow chart in appendix.
VII. Data Analysis and Discussion

We deployed our project on Amazon Elastic Compute Cloud (EC2). The instance type is t2.micro with high frequency Intel Xeon processors operating at 2.5GHz with turbo up to 3.3GHz. The workload is generated on an Intel Core i7 processor operating at 2.0GHz with 8GB RAM. Both machines run Ubuntu operating system.

Experiments are conducted with different concurrency level. Experiments of each concurrency level is repeated for ten times. As shown in the appendix section, the Apachebench returns detailed simulation result, while we mainly take two metrics into consideration: response time and requests per second. Request per second is calculated as total number of requests divided by total test time. The total test time might also include interval time if multiple request groups are sent, while response time only includes the connecting time and request processing time. The two metrics can provide basic performance measurement for our web service. We have conducted experiments on both Amazon EC2-host web service and localhost web service.

Figure 7.1 shows the relationship of concurrency and response time for EC2-host RESTful web service. Concurrency level is measure as number of requests sent simultaneously. Response time is measured in milliseconds. We can see that with increasing concurrency the response time also increases linearly, which supports our hypothesis. We have also conducted regression on concurrency and response time. Table 7.1 shows the regression result. The multiple R is 0.923, which indicates a strong linear relationship. The coefficients for intercept and x are significant: 74.32 and 0.63 respectively. We can conclude that under concurrency level from 10 to 80.
Table 7.1: Regression Result on Concurrency Level and Response Time

Figure 7.2 shows how requests per second changes with increasing concurrency level. When concurrency level is lower than 30, the number of requests per second increases because the server has not been fully used. Then after concurrency level 40 the number of requests per second decreases and becomes relatively stable afterwards. The trend also indicates good scalability of the web service under these concurrency levels.

We have also tested with higher concurrency levels for EC2-host web service, however, the result is not stable with high standard deviation. One reason is that the web service is based on cloud Amazon platform and we are using the free basic service model that might have limitation on connection such as number of simultaneous request. Therefore we also conduct performance test with localhost web service for higher concurrency level. In this case the effect of network latency would also be eliminated.
Figure 7.3a: Response Time for Localhost RESTful Web Service

Figure 7.3b: Response Time for Localhost RESTful Web Service

Table 7.2: Regression Result on Concurrency Level and Response Time

<table>
<thead>
<tr>
<th>Regression Statistics</th>
<th>ANOVA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiple R</strong></td>
<td>0.972606131</td>
</tr>
<tr>
<td><strong>R Square</strong></td>
<td>0.945962687</td>
</tr>
<tr>
<td><strong>Adjusted R Square</strong></td>
<td>0.941050204</td>
</tr>
<tr>
<td><strong>Standard Error</strong></td>
<td>11.20934556</td>
</tr>
<tr>
<td><strong>Observations</strong></td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Coefficients</strong></th>
<th><strong>Standard Error</strong></th>
<th><strong>t Stat</strong></th>
<th><strong>P-value</strong></th>
<th><strong>Lower 95%</strong></th>
<th><strong>Upper 95%</strong></th>
<th><strong>Lower 95.0%</strong></th>
<th><strong>Upper 95.0%</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>7.994536675</td>
<td>4.5290309</td>
<td>1.765109</td>
<td>-1.97417211</td>
<td>17.96324546</td>
<td>-1.974172114</td>
<td>17.96324546</td>
</tr>
<tr>
<td>X Variable 1</td>
<td>0.158445606</td>
<td>0.011418097</td>
<td>13.87671</td>
<td>2.5785E-08</td>
<td>0.133314544</td>
<td>0.1835766668</td>
<td>0.133314544</td>
</tr>
</tbody>
</table>
From detailed result in appendix we can see that connection time is nearly zero and response time is dominated by request processing time. Figure 7.3a and 7.3b shows the relationship between response time and concurrency level for localhost RESTful web service. Since the pattern is different for concurrency level below 100 and above 100, we have plotted two figures. Figure 7.3a shows that the linear relationship is strong when concurrency level is below 100, while according to Figure 7.3b, the relationship is not as stable as in low concurrency cases. However, as the regression result shown in Table 7.2, the multiple R is 0.97 and the linear correlation is still strong. Again this supports our hypothesis.
Figure 7.4a and 7.4b shows the relationship between requests per second and concurrency level. Again we used two figures for concurrency level below and above 100. When concurrency level is low and the server is not fully utilized, the number of requests per second increases when concurrency level increase. After concurrency level increases above 100, the number of requests per second keeps relatively stable from concurrency level 100 to 600 and decreases when concurrency level increases above 700.

Multiple reasons account for the decrease in number of requests per second. First, when we generate high volume of simultaneous requests, the memory might be overloaded and the system starts swapping into the paging files, this is especially likely as we generate and process request on the same machine, so that when request volume is high, the performance of web service will naturally decrease.

Second and more important, the Apachebench is a single-threaded program. Since it specifies the waiting time of requests, it is likely that Apachebench puts requests in a queue for waiting. Without paralleling Apachebench instance, the requests are not really concurrent. The deviation might be negligible at low concurrency level, however, when concurrency level is high, the error leads to a less accurate result. Specifically, since the number of requests per second is calculated as total number of requests divided by total test time, the longer than real test time (including requests queueing time) results in a lower number of requests.

VII. Conclusion and Recommendation

RESTful Web Service is a lightweight architecture. Implemented with Node.js and MongoDB, it is more suitable for real-time Web applications. Our performance test of EC2-host RESTful web service and local-host web service provides insightful information about RESTful web service performance. Although the measurement mainly focuses on software architecture bottleneck, the results supports our hypothesis of linear relationship between concurrency level and response time, which indicates good scalability of our RESTful web service.

Although REST architecture implemented with Node.js is efficient and scalable, the best implementation of web service highly depends on the function of a web-based application. Since Node.js is synchronized, a callback function is continuously monitoring I/O signal. If for a certain application, a single general request involve complicated logical flow, the procedure will be disrupted by incoming input, which might cause potential troubles in development and runtime, so that this mechanism is not suitable. Generally, the asynchronous mechanism is more suitable for real-time Web application with high concurrent requests and simple logic flow.
IX. Bibliography:


X. Appendix

Figure 10.1 Program Flow Chart
Output for Concurrent Level 100

Server Hostname: localhost
Server Port: 3000

Document Path: /search
Document Length: 2 bytes

Concurrency Level: 100
Time taken for tests: 0.294 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 246000 bytes
Total POSTed: 166000
HTML transferred: 2000 bytes
Requests per second: 3396.89 [#/sec] (mean)
Time per request: 29.439 [ms] (mean)
Time per request: 0.294 [ms] (mean, across all concurrent requests)
Transfer rate: 816.05 [Kbytes/sec] received
550.67 kb/s sent
1366.72 kb/s total

Connection Times (ms)
min mean[+/sd] median max
Connect: 0 0 0.8 0 4
Processing: 18 28 6.0 27 48
Waiting: 18 28 6.0 27 48
Total: 18 29 5.9 28 48

Percentage of the requests served within a certain time (ms)
50% 28
66% 30
75% 32
80% 33
90% 37
95% 41
98% 44
99% 45
100% 48 (longest request)

Figure 10.2: Sample Apachebench Output
<table>
<thead>
<tr>
<th>Concurrency: 10</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>113.89</td>
<td>118.34</td>
<td>123.09</td>
<td>120.69</td>
<td>120.05</td>
<td>125.04</td>
<td>118.92</td>
<td>117.14</td>
<td>97.18</td>
<td>114.84</td>
<td>116.92</td>
<td>7.33</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>35</td>
<td>37</td>
<td>36</td>
<td>36</td>
<td>35</td>
<td>35</td>
<td>37</td>
<td>35</td>
<td>43</td>
<td>43</td>
<td>37.20</td>
<td>2.99</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>48</td>
<td>45</td>
<td>41</td>
<td>42</td>
<td>45</td>
<td>42</td>
<td>41</td>
<td>45</td>
<td>56</td>
<td>39</td>
<td>44.40</td>
<td>4.61</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>83</td>
<td>82</td>
<td>76</td>
<td>78</td>
<td>80</td>
<td>76</td>
<td>78</td>
<td>80</td>
<td>99</td>
<td>82</td>
<td>81.40</td>
<td>6.31</td>
</tr>
<tr>
<td>Concurrency: 20</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>204.14</td>
<td>211.12</td>
<td>199.16</td>
<td>197.93</td>
<td>212.66</td>
<td>219.02</td>
<td>211.66</td>
<td>215.53</td>
<td>189.97</td>
<td>215.88</td>
<td>207.71</td>
<td>8.97</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>38</td>
<td>37</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>37</td>
<td>38</td>
<td>38</td>
<td>37.80</td>
<td>0.40</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>50</td>
<td>48</td>
<td>52</td>
<td>56</td>
<td>49</td>
<td>43</td>
<td>50</td>
<td>45</td>
<td>59</td>
<td>47</td>
<td>49.90</td>
<td>4.57</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>87</td>
<td>86</td>
<td>90</td>
<td>94</td>
<td>87</td>
<td>82</td>
<td>88</td>
<td>83</td>
<td>96</td>
<td>86</td>
<td>87.90</td>
<td>4.18</td>
</tr>
<tr>
<td>Concurrency: 30</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>301.24</td>
<td>299.04</td>
<td>276.88</td>
<td>297.14</td>
<td>302.91</td>
<td>299.77</td>
<td>272.1</td>
<td>268.7</td>
<td>267.44</td>
<td>295.93</td>
<td>288.12</td>
<td>14.06</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>40</td>
<td>49</td>
<td>40</td>
<td>41</td>
<td>40</td>
<td>41</td>
<td>43</td>
<td>45</td>
<td>45</td>
<td>40</td>
<td>42.40</td>
<td>2.91</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>46</td>
<td>48</td>
<td>52</td>
<td>48</td>
<td>48</td>
<td>49</td>
<td>51</td>
<td>53</td>
<td>53</td>
<td>49</td>
<td>49.70</td>
<td>2.28</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>87</td>
<td>88</td>
<td>92</td>
<td>89</td>
<td>88</td>
<td>89</td>
<td>94</td>
<td>98</td>
<td>99</td>
<td>89</td>
<td>91.30</td>
<td>4.10</td>
</tr>
<tr>
<td>Concurrency: 40</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>351</td>
<td>168.01</td>
<td>346.18</td>
<td>253.33</td>
<td>335.92</td>
<td>377.63</td>
<td>346.09</td>
<td>333.17</td>
<td>163.23</td>
<td>295.93</td>
<td>297.05</td>
<td>73.15</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>43</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>43</td>
<td>43</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>42.50</td>
<td>0.67</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>53</td>
<td>50</td>
<td>53</td>
<td>54</td>
<td>58</td>
<td>48</td>
<td>53</td>
<td>55</td>
<td>59</td>
<td>58</td>
<td>54.10</td>
<td>3.36</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>96</td>
<td>101</td>
<td>95</td>
<td>96</td>
<td>100</td>
<td>91</td>
<td>95</td>
<td>99</td>
<td>102</td>
<td>100</td>
<td>97.50</td>
<td>3.26</td>
</tr>
<tr>
<td>Concurrency: 50</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>149.08</td>
<td>157.33</td>
<td>149.62</td>
<td>149.74</td>
<td>408.95</td>
<td>147.93</td>
<td>158.72</td>
<td>315.22</td>
<td>158.81</td>
<td>144.71</td>
<td>194.01</td>
<td>86.73</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>44</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>47</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>45</td>
<td>45</td>
<td>44.80</td>
<td>0.98</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>56</td>
<td>60</td>
<td>78</td>
<td>60</td>
<td>54</td>
<td>62</td>
<td>65</td>
<td>57</td>
<td>60</td>
<td>60</td>
<td>61.20</td>
<td>6.32</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>100</td>
<td>105</td>
<td>123</td>
<td>105</td>
<td>101</td>
<td>107</td>
<td>109</td>
<td>100</td>
<td>105</td>
<td>105</td>
<td>106.00</td>
<td>6.32</td>
</tr>
<tr>
<td>Concurrency: 60</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>178.42</td>
<td>177.91</td>
<td>175.2</td>
<td>175.2</td>
<td>177.23</td>
<td>174.1</td>
<td>179.75</td>
<td>172.8</td>
<td>172.56</td>
<td>172.59</td>
<td>175.58</td>
<td>2.49</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>46</td>
<td>47</td>
<td>46</td>
<td>48</td>
<td>46</td>
<td>47</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>46</td>
<td>46.70</td>
<td>0.78</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>62</td>
<td>68</td>
<td>66</td>
<td>69</td>
<td>69</td>
<td>75</td>
<td>67</td>
<td>80</td>
<td>65</td>
<td>61</td>
<td>68.20</td>
<td>5.42</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>108</td>
<td>115</td>
<td>111</td>
<td>117</td>
<td>115</td>
<td>122</td>
<td>113</td>
<td>127</td>
<td>113</td>
<td>107</td>
<td>114.80</td>
<td>5.78</td>
</tr>
<tr>
<td>Concurrency: 70</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>201.39</td>
<td>204.44</td>
<td>203.28</td>
<td>200.14</td>
<td>216.7</td>
<td>207.9</td>
<td>204.05</td>
<td>203.96</td>
<td>196.55</td>
<td>212.79</td>
<td>205.12</td>
<td>5.65</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>50</td>
<td>49</td>
<td>49</td>
<td>48</td>
<td>48</td>
<td>50</td>
<td>48</td>
<td>49</td>
<td>48</td>
<td>51</td>
<td>49.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>70</td>
<td>66</td>
<td>65</td>
<td>65</td>
<td>64</td>
<td>69</td>
<td>73</td>
<td>79</td>
<td>88</td>
<td>69</td>
<td>70.80</td>
<td>7.15</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>120</td>
<td>115</td>
<td>114</td>
<td>113</td>
<td>113</td>
<td>112</td>
<td>118</td>
<td>122</td>
<td>129</td>
<td>136</td>
<td>120.00</td>
<td>7.20</td>
</tr>
<tr>
<td>Concurrency: 80</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
<td>Std</td>
</tr>
<tr>
<td>Request per Second</td>
<td>233.85</td>
<td>222.69</td>
<td>229.09</td>
<td>232.23</td>
<td>225.53</td>
<td>225.53</td>
<td>187.49</td>
<td>237.03</td>
<td>219.95</td>
<td>238.71</td>
<td>224.61</td>
<td>13.89</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>50</td>
<td>50</td>
<td>51</td>
<td>49</td>
<td>50</td>
<td>50</td>
<td>51</td>
<td>50</td>
<td>51</td>
<td>50</td>
<td>50.20</td>
<td>0.60</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>83</td>
<td>65</td>
<td>73</td>
<td>71</td>
<td>65</td>
<td>65</td>
<td>77</td>
<td>68</td>
<td>87</td>
<td>74</td>
<td>72.80</td>
<td>7.30</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>133</td>
<td>115</td>
<td>123</td>
<td>120</td>
<td>115</td>
<td>115</td>
<td>128</td>
<td>118</td>
<td>138</td>
<td>125</td>
<td>123.00</td>
<td>7.62</td>
</tr>
</tbody>
</table>

Table 10.1: Experiment Output for EC2-host RESTful Web Service
<table>
<thead>
<tr>
<th>Concurrency: 400</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>2628.04</td>
<td>3472.79</td>
<td>2976.12</td>
<td>3551.89</td>
<td>3266.08</td>
<td>3663.14</td>
<td>3453.25</td>
<td>3082.12</td>
<td>3436.28</td>
<td>3104.89</td>
<td>3263.46</td>
<td>299.50</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.30</td>
<td>0.46</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>112</td>
<td>82</td>
<td>99</td>
<td>85</td>
<td>90</td>
<td>85</td>
<td>90</td>
<td>99</td>
<td>84</td>
<td>92</td>
<td>91.80</td>
<td>8.76</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>115</td>
<td>86</td>
<td>103</td>
<td>88</td>
<td>93</td>
<td>87</td>
<td>93</td>
<td>102</td>
<td>87</td>
<td>96</td>
<td>95.00</td>
<td>8.83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency: 500</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>3388.87</td>
<td>2775.33</td>
<td>3142.39</td>
<td>3014.81</td>
<td>3273.9</td>
<td>2650.16</td>
<td>3376.42</td>
<td>3553.94</td>
<td>3437.58</td>
<td>3246.38</td>
<td>2998.31</td>
<td>716.80</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.30</td>
<td>0.46</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>96</td>
<td>95</td>
<td>98</td>
<td>98</td>
<td>99</td>
<td>95</td>
<td>103</td>
<td>98</td>
<td>99</td>
<td>94</td>
<td>97.50</td>
<td>2.50</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>100</td>
<td>98</td>
<td>101</td>
<td>101</td>
<td>103</td>
<td>99</td>
<td>107</td>
<td>102</td>
<td>102</td>
<td>98</td>
<td>101.10</td>
<td>2.55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency: 600</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>3375.19</td>
<td>3273.88</td>
<td>3229.68</td>
<td>3200.22</td>
<td>2821.14</td>
<td>2821.14</td>
<td>1286.9</td>
<td>2820.42</td>
<td>1044.77</td>
<td>3246.38</td>
<td>2700.53</td>
<td>798.77</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3.90</td>
<td>0.54</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>96</td>
<td>95</td>
<td>98</td>
<td>98</td>
<td>99</td>
<td>95</td>
<td>103</td>
<td>98</td>
<td>99</td>
<td>94</td>
<td>97.50</td>
<td>2.50</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>112</td>
<td>110</td>
<td>115</td>
<td>111</td>
<td>108</td>
<td>110</td>
<td>112</td>
<td>114</td>
<td>119</td>
<td>112</td>
<td>112.30</td>
<td>2.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency: 700</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>1117.31</td>
<td>1661.35</td>
<td>1093.5</td>
<td>1264.01</td>
<td>1001.93</td>
<td>2935.59</td>
<td>1074.12</td>
<td>1006.79</td>
<td>1693.7</td>
<td>1002.74</td>
<td>1385.10</td>
<td>572.70</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4.10</td>
<td>1.30</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>105</td>
<td>45</td>
<td>119</td>
<td>96</td>
<td>116</td>
<td>115</td>
<td>114</td>
<td>113</td>
<td>112</td>
<td>103</td>
<td>103.80</td>
<td>20.69</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>111</td>
<td>47</td>
<td>122</td>
<td>99</td>
<td>120</td>
<td>118</td>
<td>120</td>
<td>117</td>
<td>118</td>
<td>107</td>
<td>107.90</td>
<td>21.39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency: 800</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>935.49</td>
<td>957.925</td>
<td>891.05</td>
<td>855.96</td>
<td>861.62</td>
<td>1254.88</td>
<td>1055.2</td>
<td>2770.74</td>
<td>1003.83</td>
<td>1361.6</td>
<td>1194.83</td>
<td>548.85</td>
</tr>
<tr>
<td>Connecting Time (ms)</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4.60</td>
<td>0.92</td>
</tr>
<tr>
<td>Processing Time (ms)</td>
<td>115</td>
<td>112</td>
<td>115</td>
<td>125</td>
<td>109</td>
<td>113</td>
<td>116</td>
<td>127</td>
<td>116</td>
<td>115</td>
<td>116.30</td>
<td>5.27</td>
</tr>
<tr>
<td>Response Time (ms)</td>
<td>119</td>
<td>115</td>
<td>119</td>
<td>130</td>
<td>114</td>
<td>119</td>
<td>122</td>
<td>131</td>
<td>120</td>
<td>119</td>
<td>120.80</td>
<td>5.33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency Level</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request per Second</td>
<td>2654.69</td>
<td>3061.91</td>
<td>3202.26</td>
<td>2492.48</td>
<td>3381.37</td>
<td>3396.89</td>
<td>2798.55</td>
<td>2866.65</td>
<td>3263.46</td>
<td>2998.51</td>
<td>2700.52</td>
<td>1385.10</td>
<td>1194.83</td>
</tr>
<tr>
<td>Response Time</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>29</td>
<td>41</td>
<td>44</td>
<td>95</td>
<td>101.1</td>
<td>112.3</td>
<td>107.9</td>
<td>120.8</td>
</tr>
</tbody>
</table>

Table 10.2: Experiment Output for Localhost RESTful Web Service
Program Source Code
Front-End Source Code
//app/controllers/application.js

import Ember from 'ember';
var ApplicationController = Ember.ArrayController.extend({
  actions: {
    search: function(input) {
      this.transitionTo('search', {queryParams: {keyword: input}});
    },
    logout: function() {
      sessionStorage.clear();
      this.controllerFor('login').reset();
      this.transitionTo('login');
    }
  }
});

export default ApplicationController;

//app/controllers/comments.js

import Ember from 'ember';

var CommentsController = Ember.Controller.extend({
  rate: 0,
  name: function() {
    return sessionStorage.getItem('username');
  },
  note:'",
  actions: {
    submit: function() {
      var currId = this.controllerFor('detail').dishId;
      var self = this;
      $.ajax({
        url: globalAddress + 'comment/add',
        data: {
          "userName": self.get('name'),
          "dishId": currId,
          "content": self.get("note"),
          "rate": self.get('rate'),
          "date": $.datepicker.formatDate('M dd, yy', new Date())
        },
        type: 'POST',
        success: function(data) {
          //clear form
        }
      });
    }
  }
});

export default CommentsController;
```javascript
self.set('rate', 0);
self.set('note', '');
self.transitionTo('list');
}

export default CommentsController;

//app/controllers/list.js
import Ember from 'ember';
var ListController = Ember.ArrayController.extend({
 adminUser:function(){
   return sessionStorage.getItem("username");
}.property(),

deleteComment:function(id){
   $.ajax({
     url: globalAddress + 'comment/delete',
     data: {
       "_id": id
     },
     type: 'POST',
     success: function(data) {
       $('#'+id).remove();
     }
   });
},

event: function(e, obj) {
  var element = $(this);
  var id = element.data('id');
  this.sendAction('deleteComment', id);
},

export default ListController;

//app/controllers/login.js
import Ember from 'ember';
var LoginController = Ember.Controller.extend({
 username: null,
 password: null,

 reset: function() {
   this.setProperties(
```
actions:
login:function() {
    var username = this.get('username'),
        password = this.get('password');
    if((username==='admin' && password==='admin') ||
        (username==='paul' && password==='paul')) {
        sessionStorage.setItem("username",username);
        sessionStorage.setItem("loginState", true);
        this.transitionTo('home');
    }else{
        alert('You must type in the right username and password');
    }
}

export default LoginController;

//app/views/user-input.js

import Ember from 'ember';

var UserInputView = Ember.TextField.extend({
    didInsertElement: function() {
        this.$().focus();
    }
});

export default UserInputView;

//app/helpers/when-equal.js

import Ember from "ember";

export default Ember.Handlebars.makeBoundHelper(function(val1, val2, options) {
    if(val1===val2){
        return '✖';
    }
});

//app/routes/detail.js
import Ember from 'ember';

var DetailRoute = Ember.Route.extend({
    queryParams: {
        dishId: {
            refreshModel: true
        }
    },
    beforeModel: function() {
        if(!sessionStorage.getItem("loginState")){
            this.transitionTo('login');
        }
    },
    model:function(params){
        var detailData = $.ajax({
            //search the data on DB
            url: globalAddress + 'detail',
            data: {
                "id": params.dishId
            },
            type: 'POST',
            success: function(data) {
                return data;
            }
        });
        return detailData;
    }
});

export default DetailRoute;

/app/routes/home.js
import Ember from 'ember';

var DetailRoute = Ember.Route.extend({
    queryParams: {
        dishId: {
            refreshModel: true
        }
    },
    beforeModel: function() {
        if(!sessionStorage.getItem("loginState")){
            this.transitionTo('login');
        }
    }
});

export default DetailRoute;
model:function(params){
    var detailData = $.ajax({
        url: globalAddress + 'detail',
        data: {
            "id": params.dishId
        },
        type: 'POST',
        success: function(data) {
            return data;
        }
    });
    return detailData;
}

export default DetailRoute;

//app/routes/index.js
import Ember from 'ember';
var IndexRoute = Ember.Route.extend({
    beforeModel: function(transition) {
        this.transitionTo('login');
    },
});
export default IndexRoute;

//app/routes/new.js
import Ember from 'ember';
var NewRoute = Ember.Route.extend({
    beforeModel: function() {
        if(!sessionStorage.getItem("loginState")){
            this.transitionTo('login');
        }
    },
});
export default NewRoute;

//app/routes/search.js
import Ember from 'ember';
var SearchRoute = Ember.Route.extend({
    queryParams: {
        keyword: {
            refreshModel: true
        },
        category: {
            refreshModel: true
        }
    },
    renderTemplate: function() {
        this.render('search');
    },
    beforeModel: function() {
        if(!sessionStorage.getItem("loginState")){
            this.transitionTo('login');
        }
    },
    model: function(params){
        if (params.category) {
            var dataObject = {
                "category": params.category
            }
            var url = "category";
        } else {
            var dataObject = {
                "search_word": params.keyword
            }
            var url = "search"
        }
        var dataList = $.ajax({
            url: globalAddress + url,
            data: dataObject,
            type: 'POST',
            success: function(data) {
                return data;
            },
            async: false,
            error: function(xhr, status, error) {
                console.log(xhr);
                console.log(status);
                console.log(error);
            }
        });
        return dataList;
    }
})
export default SearchRoute;

//app/templates/application.hbs
<h1>{{#link-to 'home'}}<span>&#8962;<img src="/assets/images/home.png"></span>{{/link-to}}iFood
{{#link-to 'new'}}+ New{{/link-to}}</h1>
</header>
<div id="search-bar">
{{input-search action='search'}}
</div>
<div class="content">
{{outlet}}
</div>
<footer><span {{action 'logout'}}>Logout</span></footer>

//app/templates/list.hbs
{{#if model}}
{{#each}}
<div class="userdetail"{{bind-attr id=_id}}>
<span>
<span><strong>{{userName}}</strong> <span style="margin-left:2%;">{{date}}</span>
</span>
<br>
{{star-rating starRating=rate maxStars=5}}
<br>
<span>{{content}}</span>
<div {{action 'deleteComment' _id}} class='destroy'>{{when equal userName controller.adminUser}}
</div>

{{else}}
<p id="ifNoComment">There is no comments now!</p>
{{/if}}

//app/templates/login.hbs
<header>
  <h1>iFood</h1>
</header>
<div class='login-content'>
  <ul>
    <li>{{view 'user-input' placeholder='Username' value=username}}</li>
    <li>{{input type='password' placeholder='Password' value=password action='login' onEvent='keypress'}}</li>
    <li><button class='loginButton' {{action 'login'}}>login</button></li>
  </ul>
</div>
Dinner

Dessert

//app/templates/search.hbs

{{partial 'header'}}

<ul class="search-list">
    {{#each}}
        <li>
            {{#link-to 'detail' (query-params dishId=dishId)}}
                <p class="dish-name">{{dishname}}</p>
                <p class="dish-stuff">{{dishstuff}}</p>
            {{/link-to}}
            <p class="likes-number">{{star-rating starRating=rate maxStars=5}}</p>
            <img class="dish-image" {{bind-attr src='dishimage'}}/>
        </li>
    {{/each}}
</ul>

//router.js

import Ember from 'ember';

var Router = Ember.Router.extend({
location: IFoodAppENV.locationType
});

Router.map(function() {
    this.route('login');
    this.route('home');
    this.resource('home', function() {
        this.resource('category');
    });
    this.route('new', { path:'/new'});
    this.route('search', { path: '/home/search' });
    this.resource('detail', function() {
        this.resource('list');
        this.resource('comments');
    })
});

export default Router;

Server-side source code

Server.js

var express = require('express');
var model = require('./models/model');

var app = express();

app.configure(function () {
    app.use(express.logger('dev'));     /* 'default', 'short', 'tiny', 'dev' */
    app.use(express.bodyParser());
});

app.post('/search', model.search);
app.post('/category', model.category);
app.post('/detail', model.detail);
app.post('/comment/add', model.addComment);
app.post('/comment/get', model.getComment);
app.post('/comment/delete', model.deleteComment);
app.post('/newrecipe', model.addRecipe);
app.get('/test', model.test);

app.listen(3000);
console.log('Listening on port 3000...');

category.js

/**
 * @param express request object, it contains json data sent from front end app
 * @param express response object, used for sending response back to front end
 * when query success.
 */
exports.category = function(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    var category_word = req.body.category;
    if (category_word != null) {
        db.collection('dishes', function(err, collection) {
            var re = new RegExp(category_word, "gi");
            var query = {"category" : re};
            collection.find(query).toArray(function(err, items) {
                var empty_ary = [];
                if (category_word.length == 0) {
                    res.send(empty_ary);
                } else {
                    res.send(items);
                }
            });
        })
    } else {
        res.send([]);
    }
}

exports.add = function(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    var comment = req.body;
    var dishId = comment.dishId;
    comment.dishId = parseInt(dishId);
    var query = {"dishId" : comment.dishId};
    var newTotalRate = 0;

    //add to comments collection
    (function() {
        db.collection('comments', function(err, collection) {
            collection.insert(comment, {safe:true}, function(err, result) {
                if (err) {
                    console.log("error");
                    res.send({'error':"An error has occurred"});
                } else {
                    console.log('Success: ' + JSON.stringify(result[0]));
                    res.send(result[0]);
                }
            });
        });
    });
// Calculate average rate
(function() {
  db.collection('details', function(err, collection) {
    collection.find(query).toArray(function(err, items) {
      console.log(JSON.stringify(items[0]));
      var rate = items[0].rate;
      var rateTimes = items[0].rateTimes;
      comment.rate = parseInt(comment.rate);
      newTotalRate = (rate * rateTimes + comment.rate) / (rateTimes + 1);
      console.log("newTotalRate: ", newTotalRate);
      console.log("new rate: ", newTotalRate);
      items[0].rate = newTotalRate;
      items[0].rateTimes = rateTimes + 1;
      collection.update(query, items[0]);
    });
  });
});

(function() {
  db.collection('dishes', function(err, collection) {
    collection.find(query, function(err, document) {
      if (!err) {
        document.each(function(key, val) {
          if (val != null) {

            val.rate = newTotalRate;
            console.log("dishes: ", newTotalRate);
            collection.update(query, val);
          }
        });
      }
    });
  });
});

/** *
* Get all comments for a specific dish *
* @param req request contains dishId to query comments *
* @param res response to send back query results back to front end app in *
* json format *
*/
exports.get = function(req, res) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  var dishId = req.body.dishId;
  if (dishId != null) {

```javascript
db.collection('comments', function(err, collection) {
    var query = {'dishId': parseInt(dishId)};
    collection.find(query).toArray(function(err, items) {
        res.send(items);
    });
});

/**
 * Delete a specific comment
 * @param req request contains id of the comment to delete
 * @param res response to tell frontend whether the comment is deleted successfully
 */
exports.deleteComment = function(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    var id = req.body._id;
    db.collection('comments', function(err, collection) {
        collection.remove({'_id':new BSON.ObjectID(id)}, {safe:true}, function(err, result) {
            if (err) {
                res.send({'error': 'An error has occurred - ' + err});
            } else {
                console.log("result + ' document(s) deleted");
                res.send([req.body]);
            }
        });
    });
};

/****************************detail.js*************************************
/**
 * Get detail information of a dish, for example, ingredients and cooking steps
 * @param req request contains dishId to query details info of a dish
 * @param res response to send back dish detail info back to front end in
 *      json format
 */
exports.detail = function(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    var detail = req.body;
    var id = detail.id;
    if (id != null) {
        db.collection('details', function(err, collection) {
            var query = {'dishId': parseInt(id)};
            collection.find(query).toArray(function(err, items) {
                console.log('items ' + items[0]);
                res.send(items[0]);
            });
        });
    }
};
```
{ } else { 
    res.send([]);
}
}
******************************************************************************
/******************model.js***************************************************/
**************open database connection***************/
var mongo = require('mongodb');

var Server = mongo.Server;
var Db = mongo.Db;
BSON = mongo.BSONPure;

var server = new Server('localhost', 27017, {auto_reconnect: true});
db = new Db('iFood_db', server);

db.open(function(err, db) {
    if(!err) {
        console.log("Connected to 'iFood_db' database");
        db.collection('dishes', {strict:true}, function(err, collection) {
            if (err) {
                console.log("The 'iFood_db' collection doesn't exist. Creating it with sample data...");
                populateDB();
            }
        });
    }
});
******************************************************************************
end of db connection***********/

//search function from search.js
exports.search = require('./search').search;
exports.detail = require('./detail').detail;
exports.addComment = require('./comment').add;
exports.getComment = require('./comment').get;
exports.deleteComment = require('./comment').deleteComment;
exports.addRecipe = require('./new').addRecipe;
exports.category = require('./category').category;

//node js test function
exports.test = function(req, res) {
    res.send("hello world");
};

/**
 * hard coded data for populate data base
 */
var populateDB = function() {
    var dishes = require('./db/dishes').dishes;
}
exports.addRecipe = function(req, res) {
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader("Access-Control-Allow-Headers", "X-Requested-With");
    console.log(JSON.stringify(req.body));
    var dishDetail = req.body.dish;
    var dishSearch = req.body.search;
    var newDishId = 0;

    // add to recipe details collection
    var addDetails = function() {
        db.collection('details', function(err, collection) {
            if (!err) {
                collection.find({}, {"sort" : [['dishId', 'desc']]}).toArray(function(err, items) {
                    if (!err) {
                        dishDetail.dishId = items[0].dishId + 1;
                        newDishId = items[0].dishId + 1;
                        console.log("new dish id 1 " + newDishId);
                        dishDetail.rate = 0;
                        dishDetail.rateTimes = 0;
                        collection.insert(dishDetail);
                        addDishes();
                    } else {
                        console.log("error inserting dish detail");
                    }
                });
            } else {
                console.log("error finding details collection");
            }
        });
    }

    // add to recipe search collection
    var addDishes = function() {
        db.collection('dishes', function(err, collection) {
            if (!err) {
                collection.find({}, {"sort" : [['dishId', 'desc']]}).toArray(function(err, items) {
                    if (!err) {
                        dishSearch.dishId = items[0].dishId + 1;
                        newDishId = items[0].dishId + 1;
                        console.log("new dish id 2 " + newDishId);
                        dishSearch.rate = 0;
                        dishSearch.rateTimes = 0;
                        collection.insert(dishSearch);
                        console.log("new dish inserted to search collection");
                    } else {
                        console.log("error inserting dish search");
                    }
                });
            } else {
                console.log("error finding dishes collection");
            }
        });
    }
};
res.send([newDishId]);
    }
  });
});

addDetails();

/******************************************************************************/
/*******************
**search.js************************************************/

exports.search = function(req, res) {

  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  var search = req.body;
  var search_word = search.search_word;
  if (search_word != null) {
    db.collection('dishes', function(err, collection) {
      var re = new RegExp(search_word,"gi");
      var query = {"dishname" : re};
      collection.find(query).toArray(function(err, items) {
        var empty_ary = [];
        if (search_word.length == 0) {
          res.send(empty_ary);
        } else {
          res.send(items);
        }
      });
    } else {
      res.send([]);
    }
  } else {
    res.send([]);
  }

  /******************************************************************************/

  /******************************************************************************/
  /**************************search.js****************************************/
  /***
  * Search matched dishes name in mongodb, then send back to front end
  * @param req http request, contains search_word to query in dishes collection
  * @param res respose for sending back query result back to front end
  */

  exports.search = function(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "X-Requested-With");
    var search = req.body;
    var search_word = search.search_word;
    if (search_word != null) {
      db.collection('dishes', function(err, collection) {
        var re = new RegExp(search_word,"gi");
        var query = {"dishname" : re};
        collection.find(query).toArray(function(err, items) {
          var empty_ary = [];
          if (search_word.length == 0) {
            res.send(empty_ary);
          } else {
            res.send(items);
          }
        });
      } else {
        res.send([]);
      }
    } else {
      res.send([]);
    }
  }

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /*******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /*******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /*******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /******************************************************************************/

  /**************************************************************************)
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td><code>{ [0: {  _id: &quot;53f648a4f4f76b49a73dfdd5&quot;  dishId: 2  dishname: &quot;Sesame Beef&quot;  dishstuff: &quot;steak, garlic, vegetable oil&quot;  rate: 4.666666666666667  category: &quot;dinner&quot;  dishimage: &quot;/assets/images/beefid1.jpg&quot; } ] }</code></td>
</tr>
<tr>
<td><strong>Get Category</strong> API</td>
<td><code>http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/category</code></td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td><code>{&quot;category&quot;: &quot;breakfast&quot;}</code></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td><code>[ [0: {  _id: &quot;53f648a4f4f76b49a73dfde3&quot;  dishId: 24  dishname: &quot;Sopapilla Cheesecake&quot;  dishstuff: &quot;finely ground almonds' sugar, sour cream, white sugar&quot;  rate: 4.090909090909091  category: &quot;breakfast&quot;  dishimage: &quot;/assets/images/sopapilla_cheesecake.jpg&quot; } ] - 1: {  _id: &quot;53f648a4f4f76b49a73dfdf4&quot;  dishId: 41  rate: 0  dishname: &quot;Potato Enchiladas&quot;  dishimage: &quot;/assets/images/potato_enchiladas.jpg&quot;  category: &quot;breakfast&quot;  dishstuff: &quot;sweet potatoes, tomatoes, corn tortillas, baby spinach&quot; } ] - 2: {  _id: &quot;53f648a4f4f76b49a73dfdfa&quot; } ]</code></td>
</tr>
</tbody>
</table>
dishId: 47
dishname: "Chicken Salad"
dishimage: "assets/images/ChickenSalad.jpg"
category: "breakfast"
dishstuff: "plain lowfat yogurt, chopped red onion"
}
-
3:
{
  _id: "53f648a4f4f76b49a73dfdfb"
dishId: 48
dishname: "Basil Garlic Green Beans"
dishimage: "assets/images/BasilGarlicGreenBeans.jpg"
category: "breakfast"
dishstuff: "green beans, butter"
}
-
4:
{
  _id: "53f648a4f4f76b49a73dfdfc"
dishId: 49
dishname: "Shrimp with Dipping Sauce"
dishimage: "assets/images/ShrimpwithRemoulade.jpg"
category: "breakfast"
dishstuff: "mayonnaise, lemon juice, Shrimp"
}
-
5:
{
  _id: "53f648a4f4f76b49a73dfdd7"
dishId: 4
dishname: "Honey Chicken Kabobs"
dishstuff: "boneless Chicken, chili powder, honey"
rate: 3.7142857142857144
category: "breakfast"
dishimage: "assets/images/chickenid4.jpg"
}
-
6:
dishId: 8
dishname: "Oven-Roasted Asparagus"
dishstuff: "asparagus spears, trimmed, parmesan cheese"
rate: 5
category: "breakfast"
dishimage: "assets/images/asparagusid8.jpg"
}
- 7:
  {  
    _id: "53f648a4f4f76b49a73dfddd"
    dishId: 10
dishname: "Southwest Chipotle Salad"
dishstuff: "pasta, tomatoes, port, onions"
    rate: 2.5
category: "breakfast"
dishimage: "assets/images/mzkc.jpg"
  }
- 8:
  {  
    _id: "53f648a4f4f76b49a73dfdde"
    dishId: 11
dishname: "Bourbon Glazed Carrots"
dishstuff: "carrots, butter, bourbon"
    rate: 4
category: "breakfast"
dishimage: "assets/images/carrotid55.jpg"
  }
- 9:
  {  
    _id: "53fb940a6e1f2bd06b95e274"
    dishname: "tt2"
dishimage: ""
dishstuff: "qw, "
category: "breakfast"
    dishId: 59
    rate: 0
  }
- 10:
{  
  _id: "540563c622f9e69523ed6e0e"
  dishname: "paul test"
  dishimage: "http://timenewsfeed.files.wordpress.com/2013/02/pancakeh.jpg?w=753"
  dishstuff: "sugar, power, honey, "
  category: "breakfast"
  dishId: 62
  rate: 4
}

Get Details

API   http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/detail
Input   {"id" : 24}
Output   
  {  
    _id: "53f646eaf4f76b49a73dfdba"
    dishId: 24
    rate: 4.090909090909091
    dishName: "Sopapilla Cheesecake"
    rateTimes: 11
    category: "breakfast"
    ingredients: 
      [  
        0:  
        {  
          i-name: "cream cheese, softened"
          i-count: "1 package"
        }  
      ]  
      1:  
      {  
        i-name: "white sugar"
        i-count: "1 cup"
      }  
      2:  
      {  
        i-name: "vanilla extract"
        i-count: "1 teaspoon"
      }  
      3:  
      {  
        i-name: "refrigerated crescent rolls"
      }
    }
}
"Preheat an oven to 350 degrees F (175 degrees C). Prepare a 9x13 inch baking dish with cooking spray."

"Beat the cream cheese with 1 cup of sugar and the vanilla extract in a bowl until smooth; set aside."

"Unroll the cans of crescent roll dough, and use a rolling pin to shape each sheet into 9x13 inch rectangles. Press one sheet into the bottom of a 9x13 inch baking dish. Evenly spread the cream cheese mixture into the baking dish, then cover with the remaining piece of crescent dough, and brush with melted butter. Stir together 1/2 cup of sugar with the cinnamon in a small bowl, and sprinkle evenly over the top of the cheesecake."

"Bake in the preheated oven until the crescent dough has puffed and turned golden brown, about 30 minutes. Allow to cool 10 minutes before serving."
userName: "Tina"
dishId: 24
content: "I like it!"
rate: "5"
date: "Aug 24, 2014"

Add Comment
API  http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/comment/add
Input  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>{ &quot;userName&quot;: &quot;admin&quot;,</td>
</tr>
<tr>
<td>2.</td>
<td>&quot;dishId&quot;: &quot;24&quot;,</td>
</tr>
<tr>
<td>3.</td>
<td>&quot;content&quot;: &quot;I like this dish&quot;,</td>
</tr>
<tr>
<td>4.</td>
<td>&quot;rate&quot;: &quot;3&quot;,</td>
</tr>
<tr>
<td>5.</td>
<td>&quot;date&quot;: &quot;Sep 02, 2014&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Output  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>{</td>
</tr>
<tr>
<td>2.</td>
<td>userName: &quot;admin&quot;</td>
</tr>
<tr>
<td>3.</td>
<td>dishId: 24</td>
</tr>
<tr>
<td>4.</td>
<td>content: &quot;I like this dish&quot;</td>
</tr>
<tr>
<td>5.</td>
<td>rate: &quot;3&quot;</td>
</tr>
<tr>
<td>6.</td>
<td>date: &quot;Sep 02, 2014&quot;</td>
</tr>
<tr>
<td>7.</td>
<td>_id: &quot;540583c822f9e69523ed6e13&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Delete Comment
API  http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/comment/add
Input  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>{ _id: &quot;540583c822f9e69523ed6e13&quot; }</td>
</tr>
</tbody>
</table>

Output  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>{</td>
</tr>
<tr>
<td>2.</td>
<td>userName: &quot;admin&quot;</td>
</tr>
<tr>
<td>3.</td>
<td>dishId: 24</td>
</tr>
<tr>
<td>4.</td>
<td>content: &quot;I like this dish&quot;</td>
</tr>
<tr>
<td>5.</td>
<td>rate: &quot;3&quot;</td>
</tr>
<tr>
<td>6.</td>
<td>date: &quot;Sep 02, 2014&quot;</td>
</tr>
<tr>
<td>7.</td>
<td>_id: &quot;540583c822f9e69523ed6e13&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

New Recipe
API  http://ec2-54-191-21-124.us-west-2.compute.amazonaws.com:3000/newrecipe
Input  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>{ &quot;dish&quot;: { &quot;dishName&quot;: &quot;steak&quot;, &quot;imageUrl&quot;: &quot;<a href="http://www.trbimg.com/img-50743ff9/turbine/os-best-steak-restaurants-orlando-">http://www.trbimg.com/img-50743ff9/turbine/os-best-steak-restaurants-orlando-</a>&quot; }</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>[1]</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>0:</td>
<td>64</td>
</tr>
</tbody>
</table>