CS291A: SCALABLE INTERNET SERVICES - FALL 2021

PRIMARY PROJECT REPORT

eKirana

GitHub - https://github.com/scalableinternetservices/HackOverflow

---

HackOverflow Members:
Navya Battula
Shubham Talbar
Arjun Prakash
Sharath Chandra Vemula
Saikumar Yadugiri

Instructrors:
Dr. Bryce Boe
Dr. Nevena Golubovic

December 3, 2021

Contents

1 Introduction

The University of California Santa Barbara is a bright and vibrant university campus. The University and its surroundings offer many lively places to eat, shop, and surf. But with the growing international population, these places are not enough to serve a diverse diaspora. The Indian community has been growing steadily over the last few years due to the increased International intake of Graduate degree students. There is a dearth of Indian restaurants and grocery places in and around Isla Vista, Goleta, and Santa Barbara. Currently, there is a local vendor who delivers Indian groceries to San Clemente Villages from Los Angeles every fifteen days. The vendor has to manually collect orders from every student over a Whatsapp group. As students of the Computer Science department, we felt the need to automate this order collection process and provide such local vendors the ability to serve the bubbling Indian (or any other) community of Santa Barbara. Hence, we created **eKirana**, an online e-commerce website that serves the purpose of connecting local vendors with a specific community. The word Kirana refers to local gully shops in India. The idea is to provide an e-platform to such businesses so that they can market their products to a larger audience. The technology stack used for building the e-commerce site is Ruby on Rails. We also performed load testing on the site using Tsung. This report will highlight the features offered by the web application followed by the nitty-gritty of the implementation. In the end, we also discuss the Tsung testing result for different test workflows simulating typical user activity.

2 Structure

In section 3, we provide the salient features of our website and show the various web pages we have and their GUI representations. In section 4, we provide a deep-dive into the database relations using ER diagram and explain the various models present in our Rails website framework. In section 5, we explain the probabilistic flows that our users are most likely to take and improve these flows using various optimization techniques. In section 6, we scale our website in multiple angles and find the right set of application and database instances to run a demand specified in user arrival rates. In section 7, we provide a scope of implementable improvements within a reasonable frame of time and conclude the report in section 8.

3 Features

eKirana is an e-commerce website with essential features to help sellers register themselves and facilitate buyers with placing their orders. The major goal of the project is to create a web application and stress test it through multiple scenarios to find bottlenecks. Once a few bottlenecks are identified the idea is to optimize these workflows to improve the end-to-end performance. eKirana helps buyers view various items, check their ratings, add them to a cart, and order all items in a cart in bulk. The platform also allows potential sellers to register themselves and list products/items that they want to sell. Both seller and buyer views have a profile view where we highlight important information about the party. The Homepage of the application lists all the available items offered by various sellers. The item page is where we display a detailed view of a particular product. Both these pages do not need a buyer to specifically log in and any unauthenticated user can access these. For this project, we have not uploaded any images for items as load testing the platform shouldn't be affected by static images. The user flow diagram and the application GUI are shown in the following subsections.

3.1 User Action Flow

Figure 1 depicts a generic user action flow for the website. To view the items listed on the website or the details of a particular product (such as its ratings), a user need not log in. But to place an order or create a new item, a user needs to register (sign up) or log in (sign in) to their respective account.

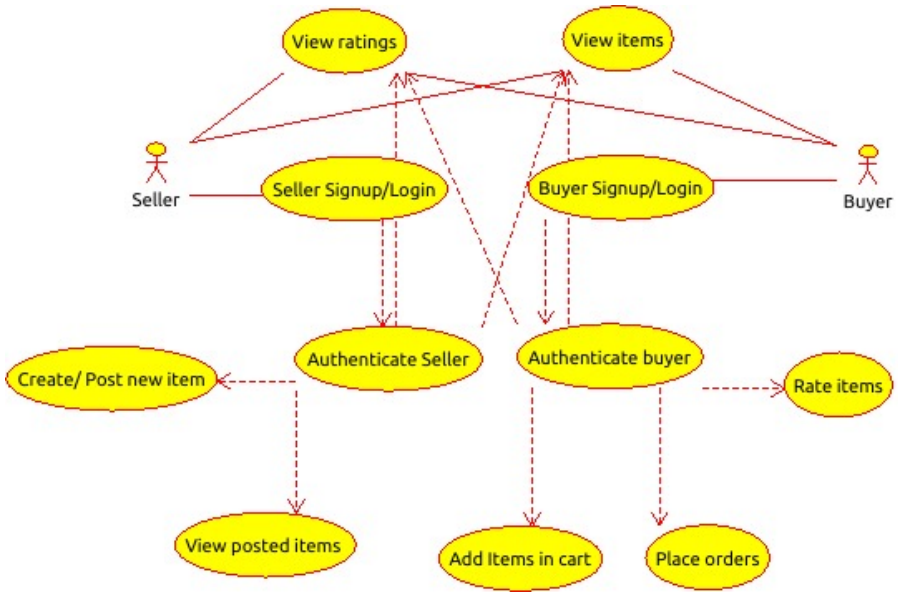
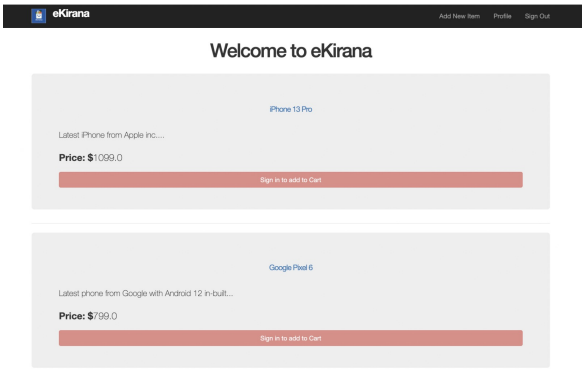


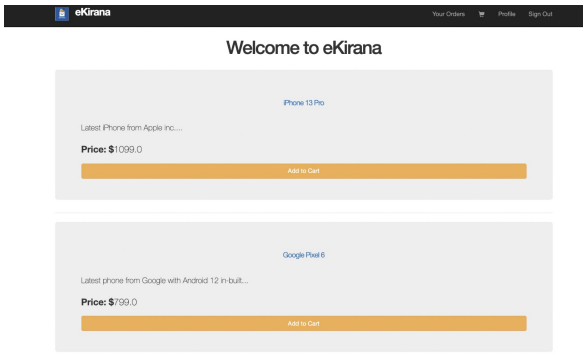
Figure 1: Generic User Action Flow.

3.2 Homepage

After a user (logged-in or new) requests the website's root URL, they will be redirected to the homepage which contains the list of all the items present in our website's database. The homepage also has links in the top navbar which provide users the ability to log in or sign up.



(a) Home Page for Unauthenticated Users



(b) Home Page for Buyers

3.3 Item Page

If a user wants to view additional details about a particular item, they can click on a particular item's link and they will be redirected to the Item Page where they can check the full description and all the user ratings. A user can also assess the rating for a particular item based on the star-based rating awarded to it by previous users. This star rating is the average of all the ratings given to the product by all the Buyers.

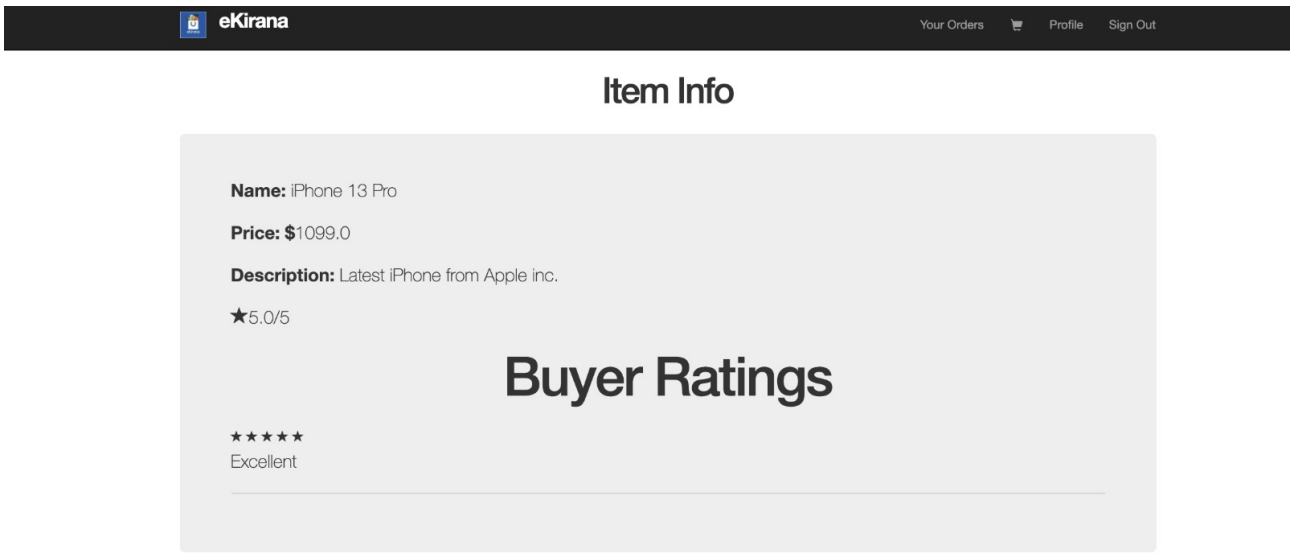
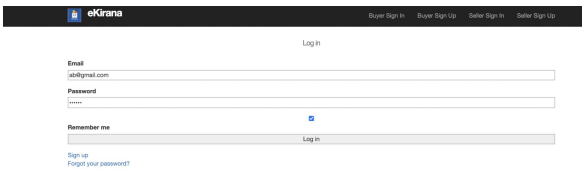


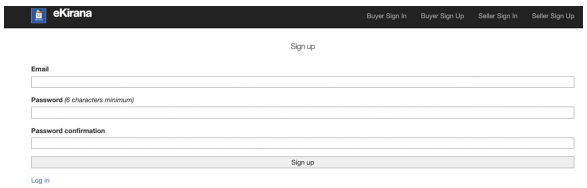
Figure 3: Item Details Page

3.4 User Authentication

The login pages for Buyer and Seller are the same. So, we are only showing the pages for the Buyer account. The only difference between these two user models is the functionality we allow for those models. Buyers as the name suggest, can add items to their carts, place orders, and view their orders. In addition, they can rate their orders, and thus, they are in-turn rating items. Sellers can add new items to the item pool offered by the site.



(a) Sign In/Log In Page Users



(b) Sign Up/Register Page for Users

3.5 Buyer Cart Page

A Buyer's cart lists all the items a user wishes to buy. We included javascript functionality to increase the quantity of each item from 1 to 10. The site also provides a delete button that will remove a particular item from the cart. We also have a javascript code to add the price of an item with the quantity of all the items in the cart and show the overall price of the cart.

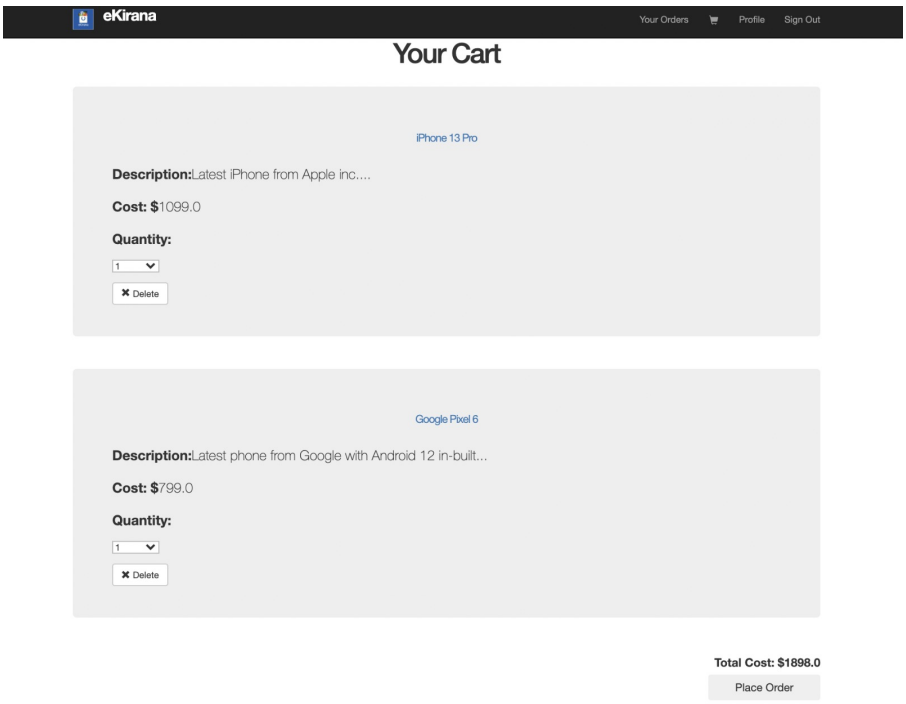


Figure 5: Cart Page

3.6 Buyer Orders Page

On a Buyer's Order page, they can view all the previous orders they have placed so far. There is information about the quantity of the item placed and a button to rate the order if it is not rated yet.

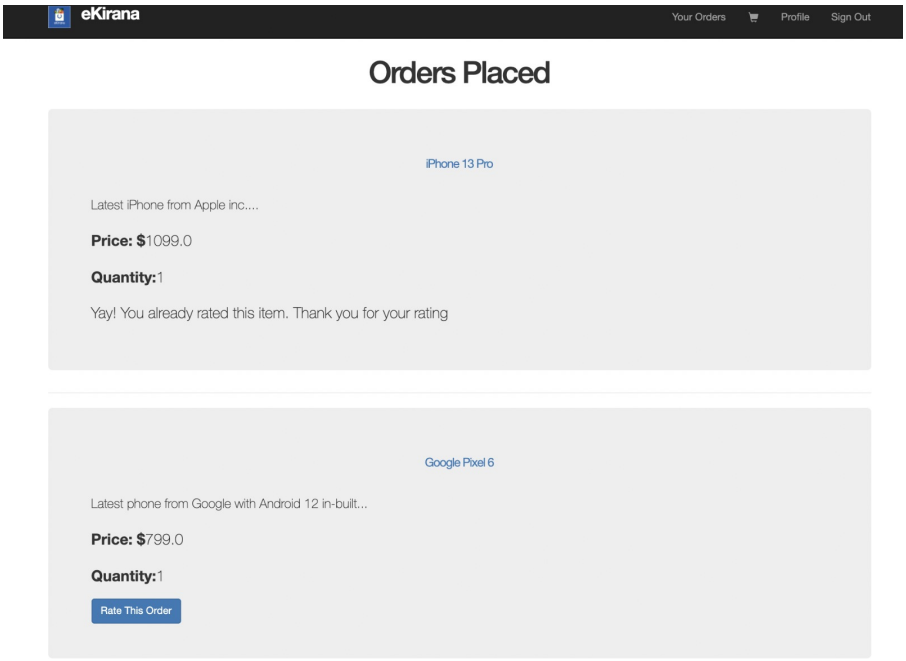


Figure 6: Orders Page

3.7 Buyer Ratings Page

Buyers can use this page to rate their orders. The order for which he is rating is shown in the heading. He can provide a numerical star rating from the drop-down menu and a comment to justify his rating.

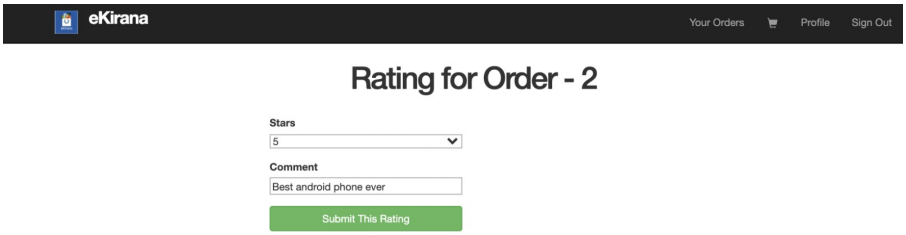


Figure 7: Ratings Page

3.8 Buyer Profile Page

The profile page for the Buyer just contains their email address and links to their cart and orders

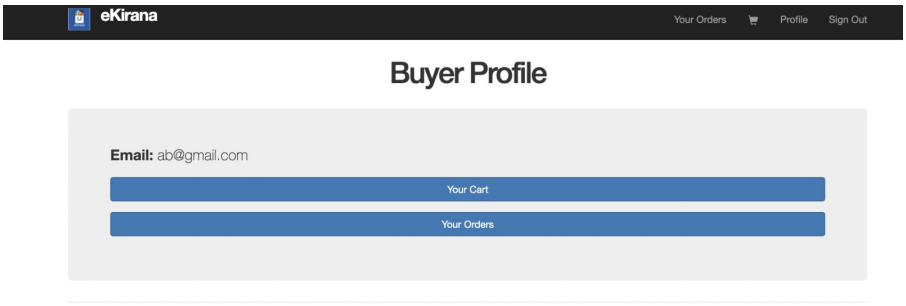


Figure 8: Buyer Profile Page

3.9 Seller Profile Page

The profile page for Seller contains their email address and links to all the items he sold so he can make sure they are currently displayed in the homepage.

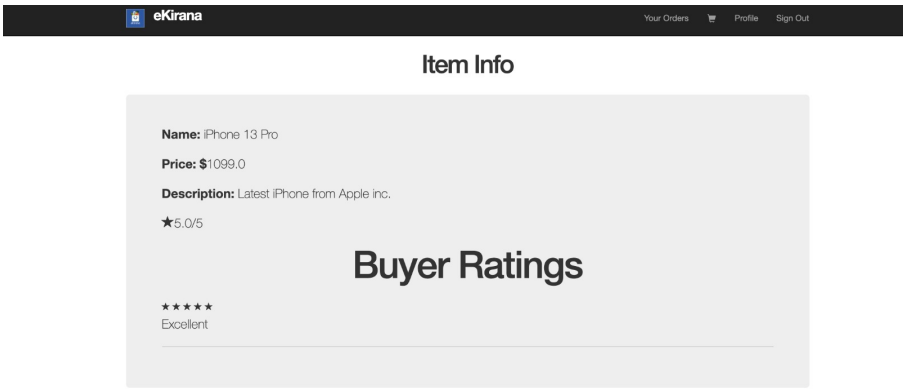


Figure 9: Seller Profile Page

3.10 New Item Page

To create a new item, the user needs to be authenticated as a Seller. Then they visit the New Item page and create a new item. This form contains a name, description, and price fields, which are the essentials of an item.

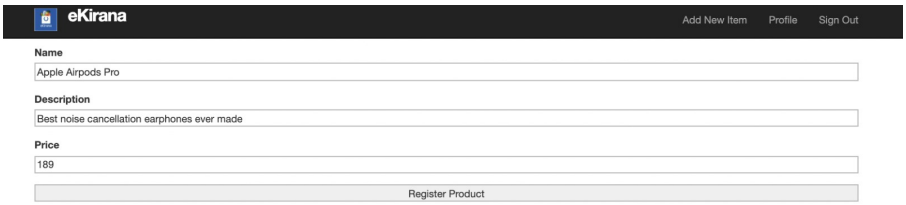


Figure 10: New Item Form

4 Models

Figure 11 below captures all the important models on the eKirana web app. The entire platform is abstracted to model different interactions and enable optimizations for different workflows.

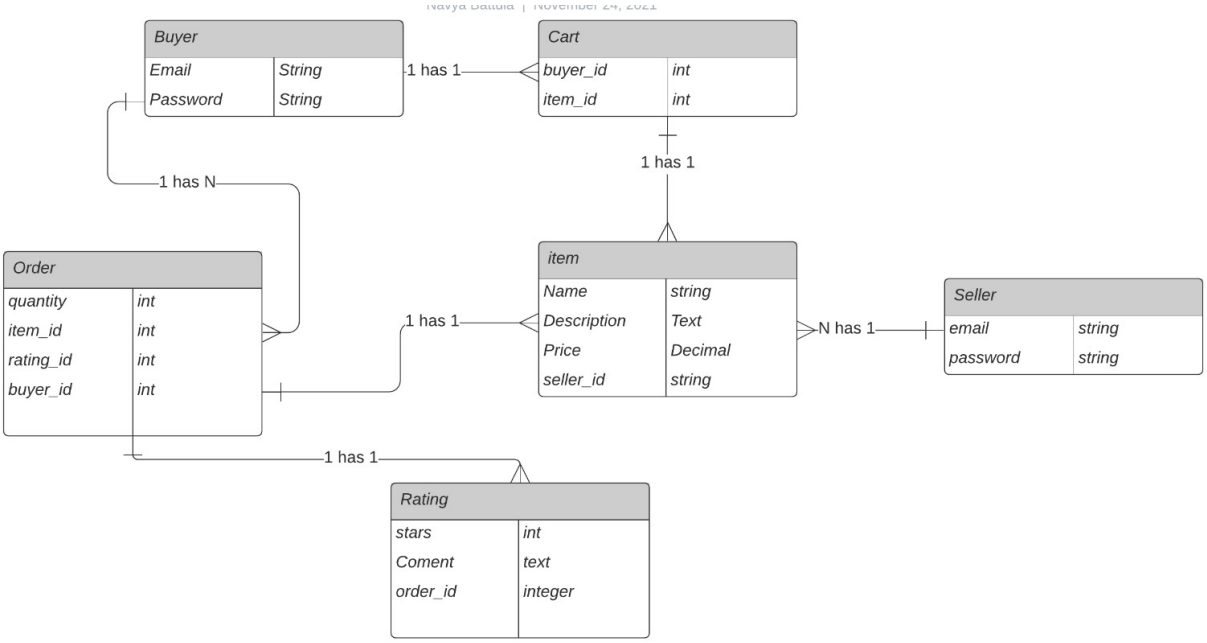


Figure 11: Entity Relationship Diagram

4.1 Item Model

The Item model contains name of the product, description of the item along with with price and ID of the seller who listed the item. The data types for each of the attribute is listed in Figure 12.

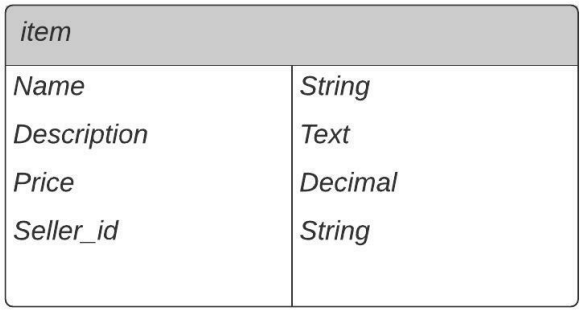


Figure 12: Item Model

4.2 Cart Model

The Cart model contains the item ID and the ID of the buyer who owns the cart. The data types for each of the attribute is listed in Figure 13.

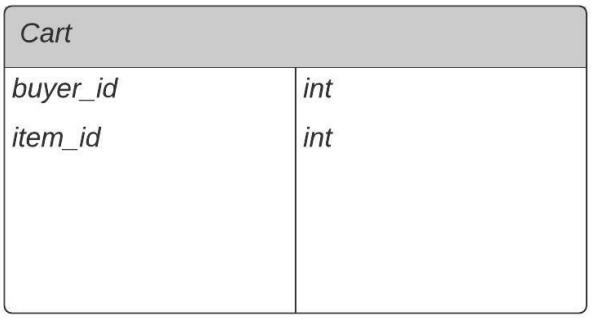


Figure 13: Cart Model

4.3 Order Model

The Order model contains the item ID, quantity of the item along with the rating ID of the order and the ID of the buyer to whom the order belongs to. The data types for each of the attribute is listed in Figure 14.

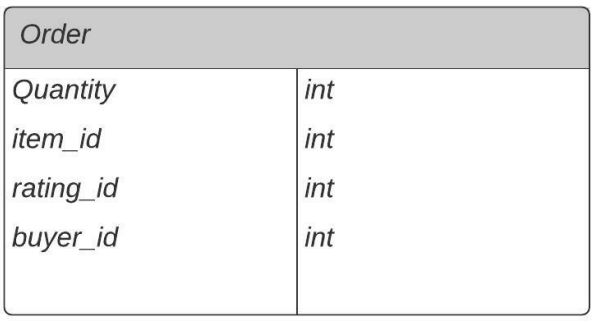


Figure 14: Order Model

4.4 Rating Model

The Rating model contains an integer number from 1 to 5 which entails the satisfaction of the buyer for a particular item. It also contains a comment attribute to signify if a buyer has a specific comment about the order and an order ID to signify the order associated with the rating.

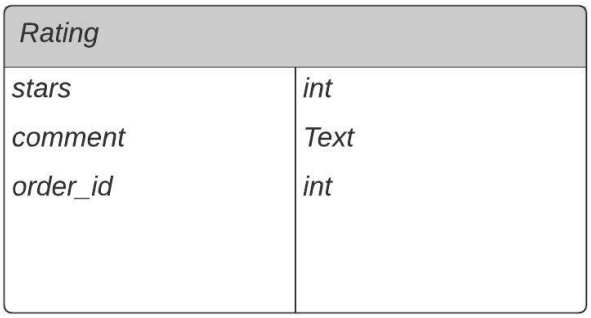


Figure 15: Rating Model

4.5 Buyer Model

The Buyer model is generated by devise gem. So, it contains many table entries like reset_password_token, encrypted_password etc which are not really pertinent to the use case. The unique identifier for buyers is their respective email addresses.

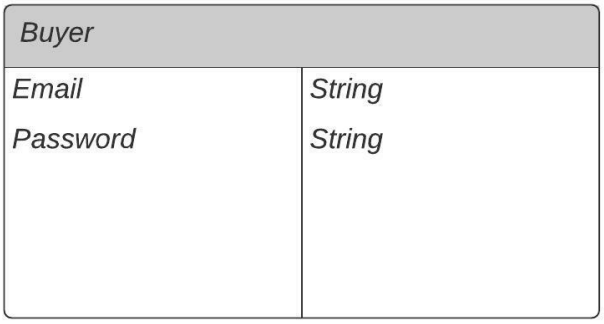


Figure 16: Buyer Model

4.6 Seller Model

The Seller model is also generated by devise gem. It contains numerous table entries like reset_password_token, encrypted_password etc which are not relevant for the use case. Sellers are uniquely identified based on their email address.

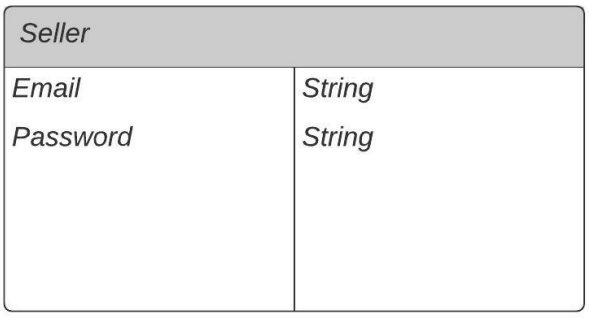


Figure 17: Seller Model

5 Workflows

In order to test the eKirana web application, we came up with several different test workflows. Every test workflow is a simulated user behaviour typical to an e-commerce platform. In total we executed five different workflows over a span of seven minuted each. Every workflow was divided in seven phases where for every consecutive phase the arrival rate incremented by a factor of two. The test scenario for every workflow is defined below.

Test Scenario setup

Instance

1. Application server: c5.large
2. Database: db.m5.large

Tsung load Total 7 phases where each phase lasts for 60 seconds.

1. **Phase 1:** 2 users/sec
2. **Phase 2:** 4 users/sec
3. **Phase 3:** 8 users/sec
4. **Phase 4:** 16 users/sec
5. **Phase 5:** 32 users/sec
6. **Phase 6:** 64 users/sec
7. **Phase 7:** 128 users/sec

5.1 Workflow 1: Buyer visits Homepage

This workflow simulates a user visiting the homepage of the application. Potential bottlenecks here could be when the total items listed by different sellers increase exponentially.

1. Buyer logs in
2. Buyer visits home page

3. Buyer logs out

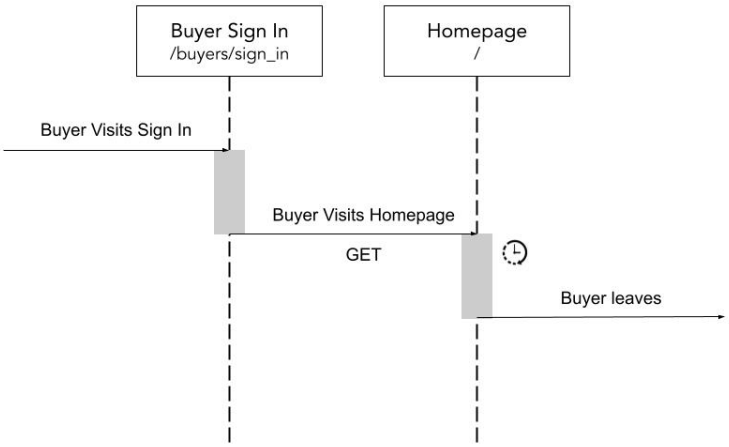


Figure 18: Workflow 1: Buyer Visits Homepage

Bottleneck Analysis We were initially getting 422 errors because of invalid CSRF token generated by devise gem. We got around the issue by adding skip_forgery_protection in application_controller.rb

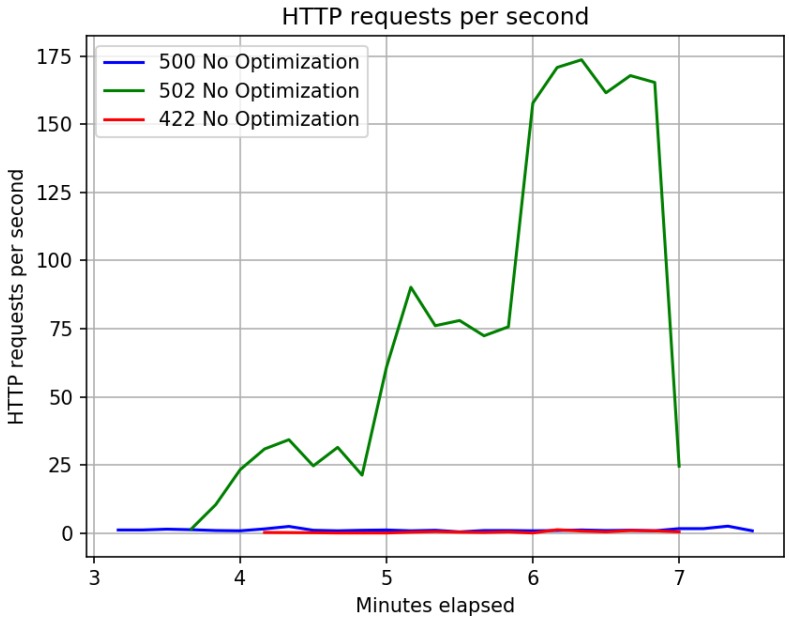
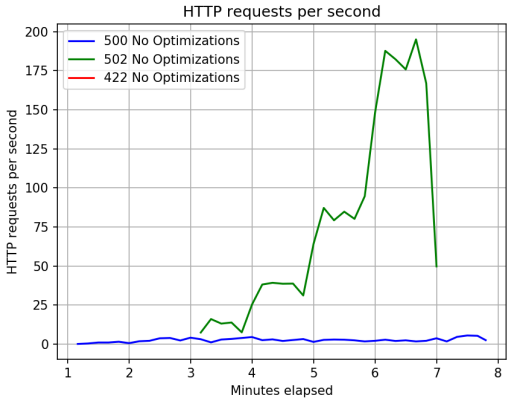
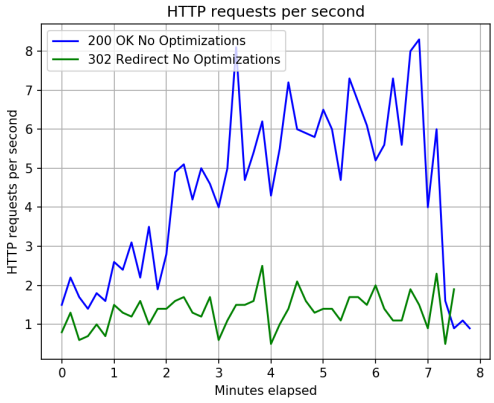


Figure 19: Initial 422 Responses

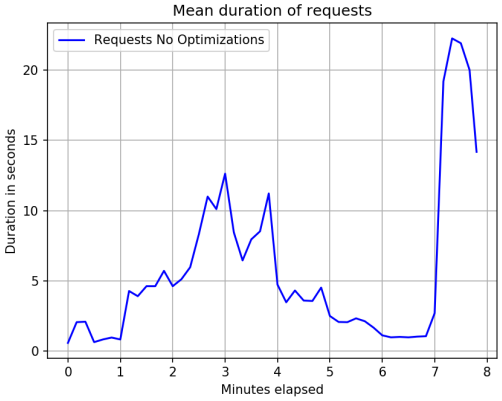
From the initial load test we observed major bottlenecks in this flow. The request mean duration was more than 20 sec in the final phase. We also noted 500 errors starting in phase 1 and 502 errors starting in phase 3.



(a) 5XX Responses



(b) 200, 300 Responses

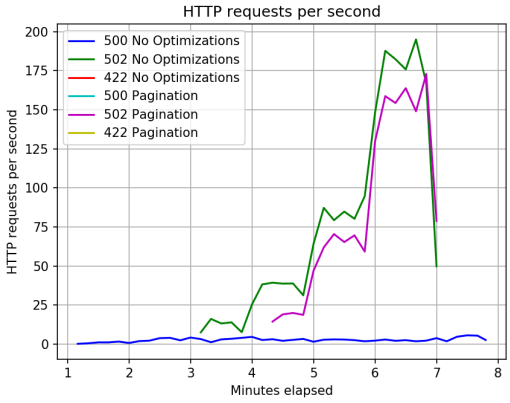


(c) Mean Request Duration

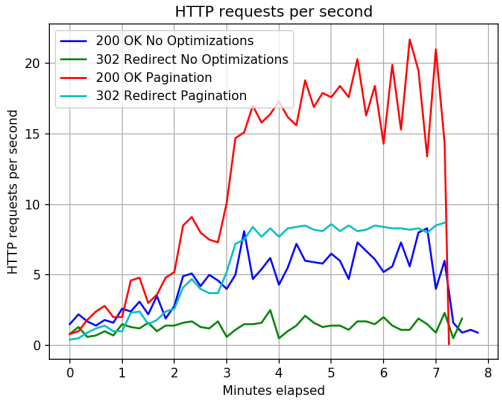
Figure 20: Initial Status Without Any Optimizations

Optimizations Below are different optimization techniques we tried and could improve the test workflow.

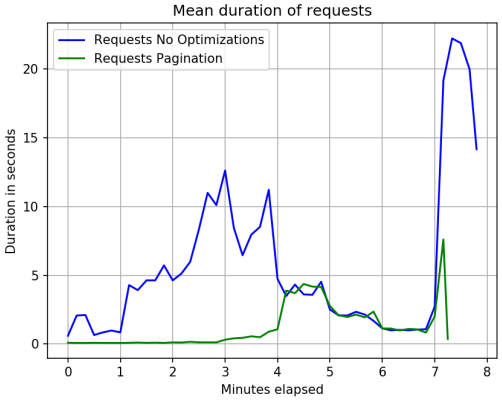
Pagination Our database is seeded with 10,000 items and the home page displays all the items initially. With pagination we reduced the number of items on each page to 10. After pagination we found significant improvement in mean duration of requests, it remained almost constant until phase 4. The max mean duration was reduced from more than 20 sec to around 7 secs. We also found the number of 502 errors reduced a bit and started only in phase 4. Number of 200 and 302 also increased in each phase compared to no optimization.



(a) 5XX Responses



(b) 200, 300 Responses



(c) Mean Request Duration

Figure 21: Workflow Status After Pagination

Caching After pagination, we tried caching and we didn't find much improvement as each page now has only 10 items. We are not putting up any graphs for it.

5.2 Workflow 2: Buyer logs in and adds items to cart

This workflow simulates the following scenario

1. Buyer logs in
2. Buyer visits home page
3. Buyer views an item, view its ratings
4. Adds an item to the cart
5. Buyer visits cart page
6. Buyer logs out

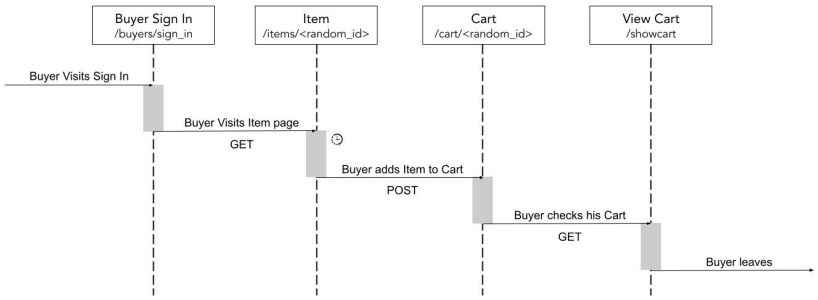
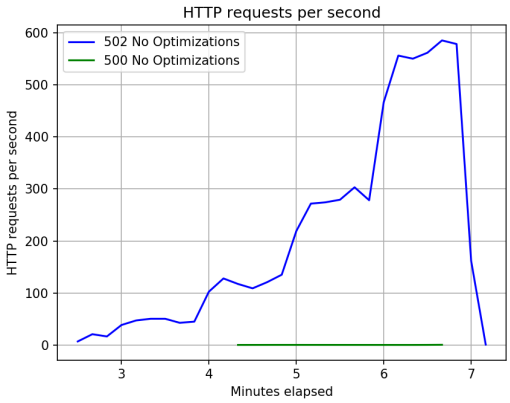
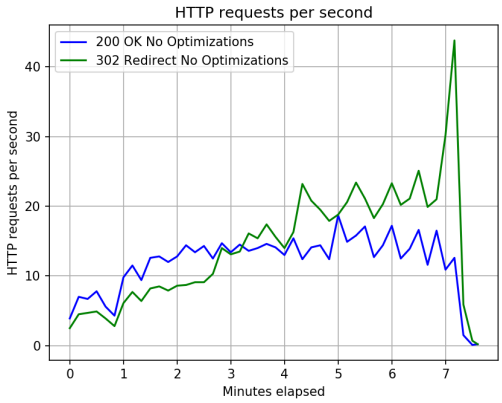


Figure 22: Workflow 1: Buyer Logs In and Adds Item to Cart

Bottleneck Analysis In the initial load testing, the 502s start occurring from phase 3 and the 200s and 302s also increase accordingly and reach peak value in phase 7. Also after phase 3 we could observe an increase in mean duration time which means that there were potential bottlenecks occurring after phase 3.



(a) 5XX Responses



(b) 200, 300 Responses

(c) Mean Request Duration

Figure 23: Initial Status Without Any Optimizations

Optimizations Below are different optimization techniques we tried and could improve the test workflow.

Pagination We used 10,000 items in our database for our application and in the initial load test we tried to load all the items at once which caused performance issues for the application. We attempted fixing it with pagination. After performing pagination and fixing the number of entries per page to 10, we observed that the mean duration time improved and the number of 502s reduced a bit. The 200s and 302s remained the same.

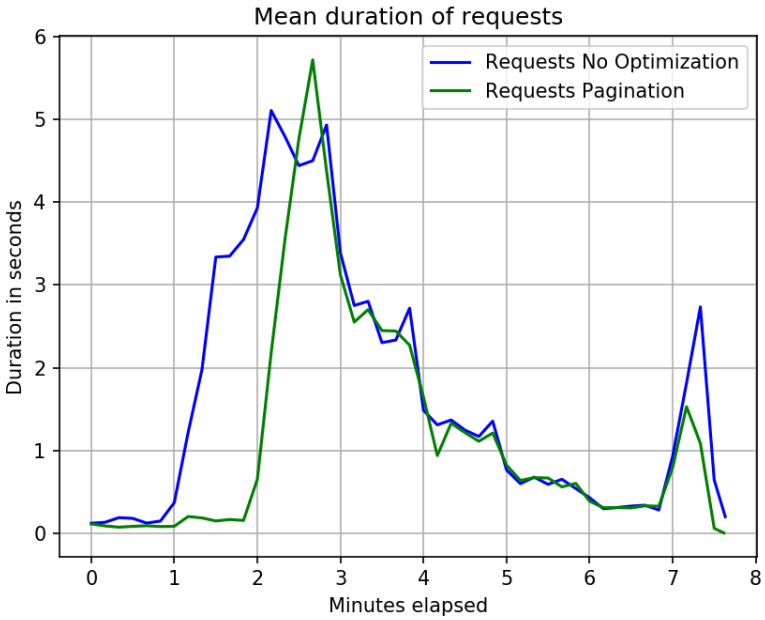
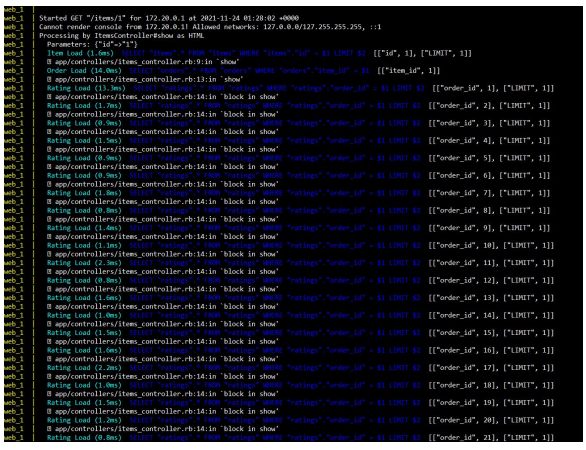


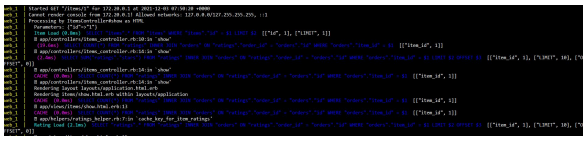
Figure 24: Workflow status after pagination

Caching After pagination, we tried Caching and we didn't find much improvement in the metrics as each page has only 10 items in it. We are not putting any graphs for this.

N+1 Query Optimization To clear N+1 Query Optimization, we didn't use includes but modified the Item model to have 1 to N relation with Rating via Order using :through keyword in Rails. So, we were able to optimize N+1 queries to 6 everytime.



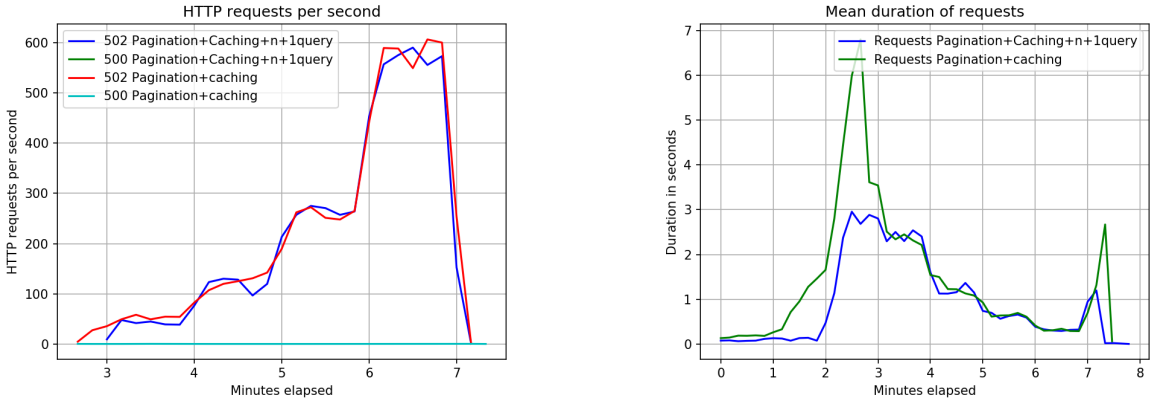
(a) Before N+1 Query Optimization



(b) After N+1 Query Optimization

Figure 25: N+1 Query Optimization for Item Model

After pagination, we tried another optimization - N+1 Query Optimization on items and rating page. We saw a significant improvement in the mean duration time (it fell below 3 seconds which is less than half of what was observed previously). The number of 502s reduced a little bit and the 200s and 302s are pretty much the same throughout.



(a) 5XX Responses

(b) Mean Request Duration

Figure 26: Workflow Status After N+1 Query Optimization

Indexing Finally we included indexing, and didn't't see any major improvement. We are not putting any graphs for this.

5.3 Workflow 3: Buyer logs in and places order

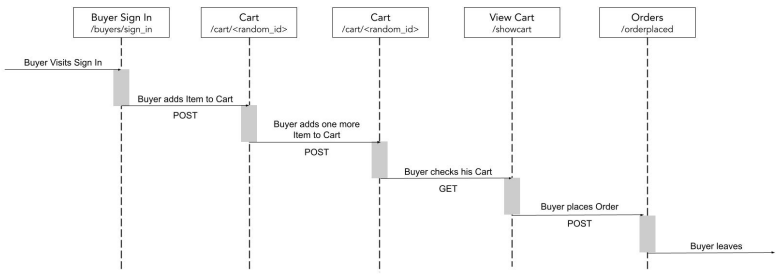


Figure 27: Workflow 3: Buyer Logs In and Places Order

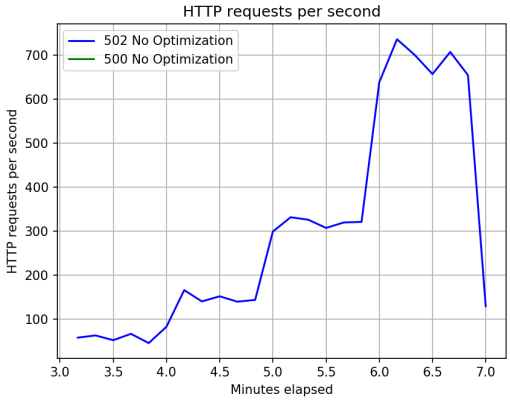
This workflow simulates the following scenario

1. Buyer logs in
2. Buyer visits home page
3. Buyer adds one item to cart
4. Buyer adds another item to cart
5. Buyer visits cart page

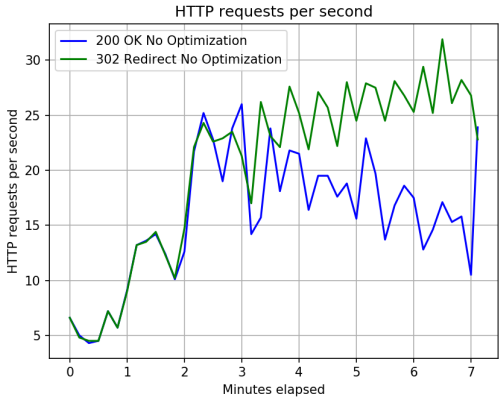
6. Buyer places his order

7. Buyer logs out

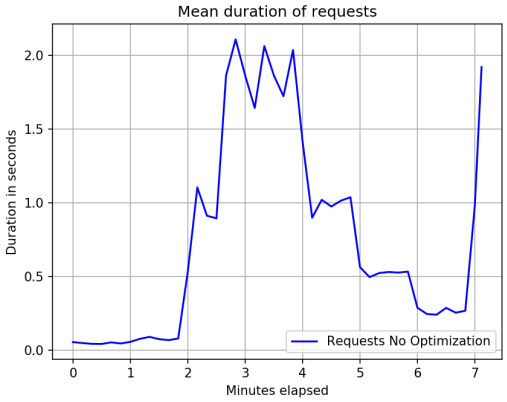
Bottleneck Analysis From the initial load test we could observe that the 502 responses start in phase 4 (i.e after 3 minutes). And we could see the 200 and 302 responses keep increasing until phase 3 and start to flatten out after that. Also, the mean duration of requests started increasing in phase 3, which indicates that there are bottlenecks starting phase 3.



(a) 5XX Responses



(b) 200, 300 Responses

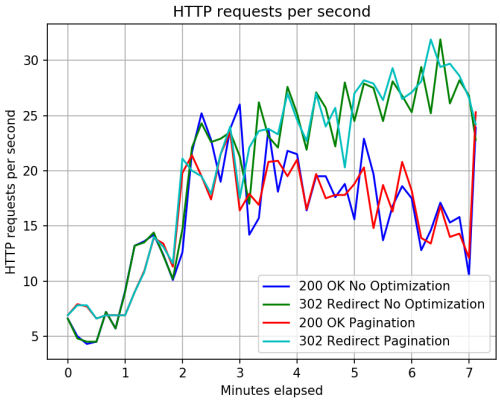


(c) Mean Request Duration

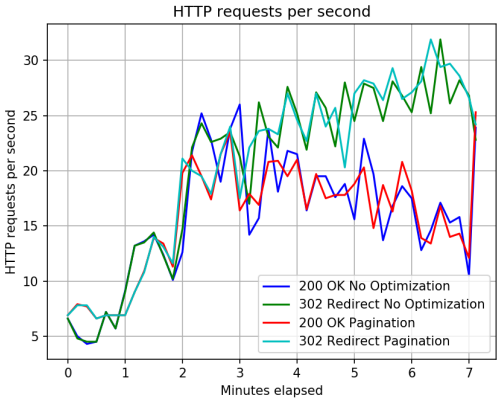
Figure 28: Initial Status Without Any Optimizations

Optimizations Below are different optimization techniques we tried and could improve the test workflow.

Pagination Our database is seeded with 10,000 items and the home page displays all the items initially. With pagination we reduced the number of items on each page to 10. After pagination we found the mean request duration was reduced to less than 1 sec from 2 sec during phase 3. Number of 200 and 302 requests remained almost the same.



(a) Mean Duration of Requests

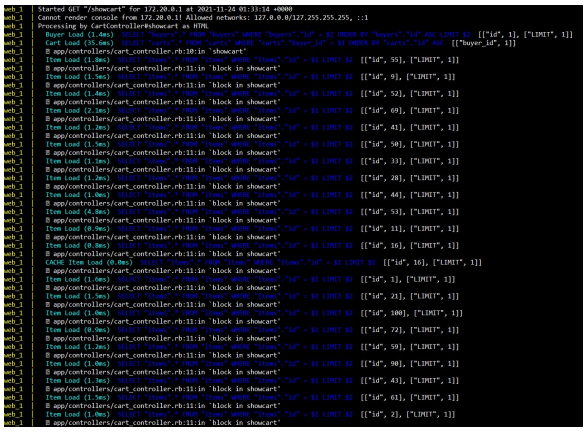


(b) 200, 300 Responses

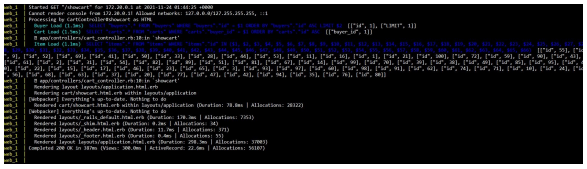
Figure 29: Workflow 3 after Pagination

Caching After pagination, we tried Caching and we didn't find much improvement in the metrics as each page has only 10 items in it. We are not putting any graphs for this.

N+1 optimization After pagination and caching, we tried N+1 optimization on the cart page, and we found not much improvement as even the cart page is paginated and has only 10 items in it. We are not putting any graphs for this.



(a) Before N+1 Query Optimization



(b) After N+1 Query Optimization

Figure 30: N+1 Query Optimization for Cart Model

Indexing Finally we included indexing, and didn't see any major improvement.

5.4 Workflow 4: Buyer rates order

This workflow simulates the following scenario:

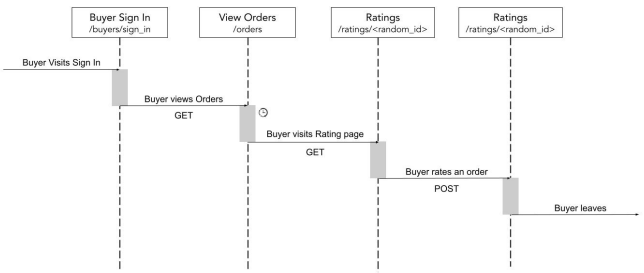
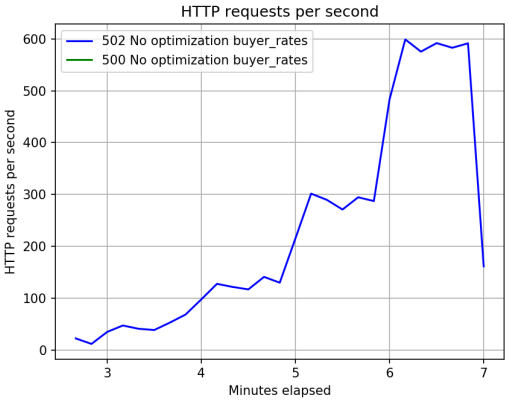


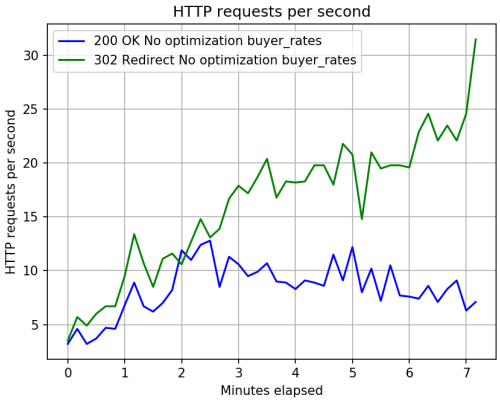
Figure 31: Workflow 4: Buyer Rates An Order

1. Buyer signs in
2. Buyer views all his orders
3. Randomly rates an item
4. Buyer leaves

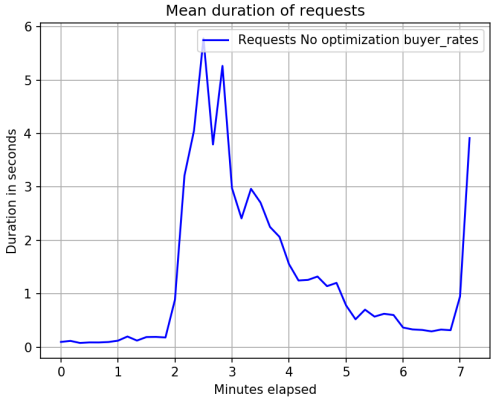
Performance Analysis Initial load test without any optimization shows that there are major bottlenecks in the home page. We can see that 502 errors start after 20 seconds (in 1st phase). Also, the 200 and 302 responses per second are also very low (30). We can see that mean duration is also very high (6 seconds) which suggests that there are a lot of improvements needed in the home page and the orders page.



(a) 5XX Responses



(b) 200, 300 Responses



(c) Mean Request Duration

Figure 32: Initial Status Without Any Optimizations

Optimizations Below are different optimization techniques we tried and could improve the test workflow.

Pagination For this workflow, pagination didn't improve the mean time much because we're not visiting the home page and the orders page doesn't have too many items to take advantage of pagination.

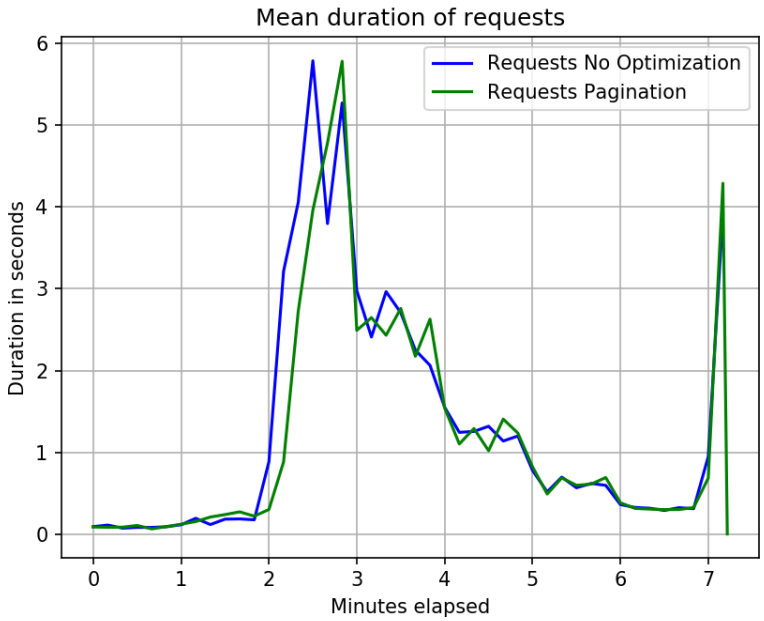
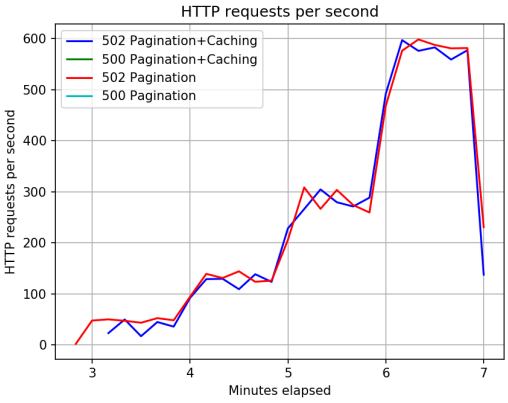
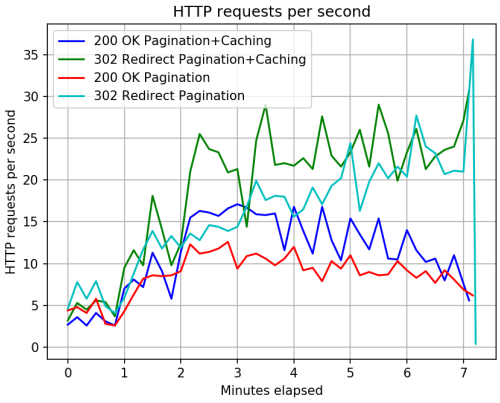


Figure 33: Mean Duration of Requests

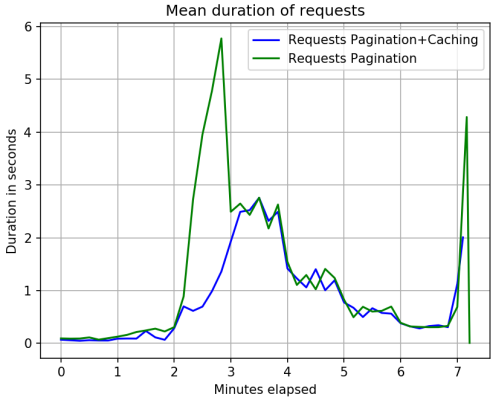
Caching The next performance optimization we did is Caching. Caching improved the performance a lot(mean time reduced by 50% and HTTP requests increased by 40%) as orders are cached for a fixed set of buyers.



(a) 5XX Responses



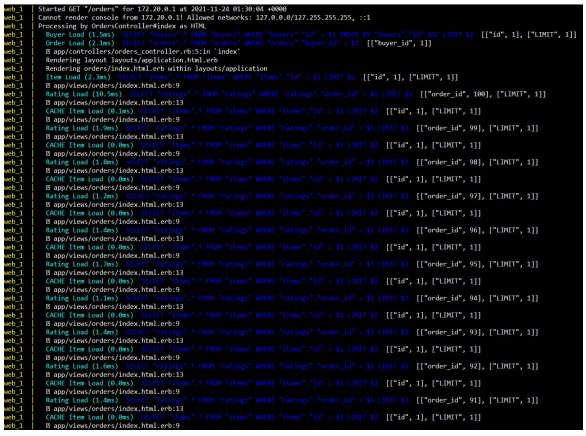
(b) 200, 300 Responses



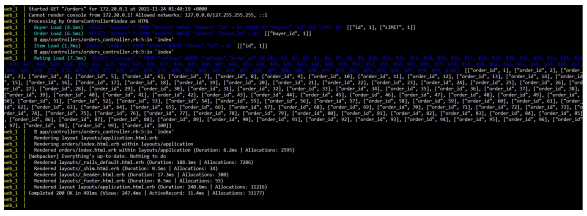
(c) Mean Request Duration

Figure 34: Workflow After Caching

Indexing and N+1 query optimisation We added indexing and N+1 query optimisation, but we didn't see any major improvements. We are not putting any graphs for this.



(a) Before N+1 Query Optimization



(b) After N+1 Query Optimization

Figure 35: N+1 Query Optimization for Order Model

5.5 Workflow 5: Seller Adds An Item

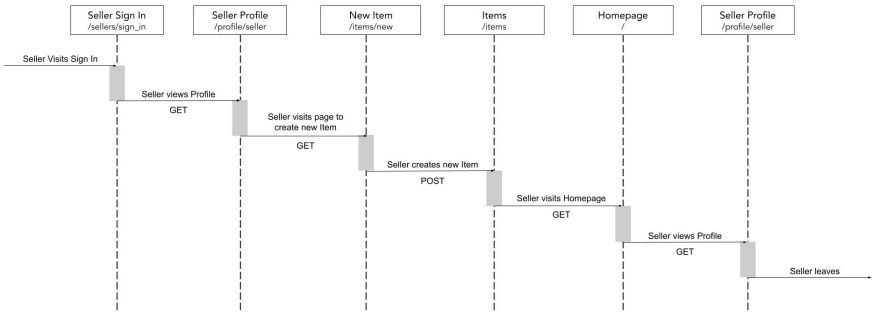
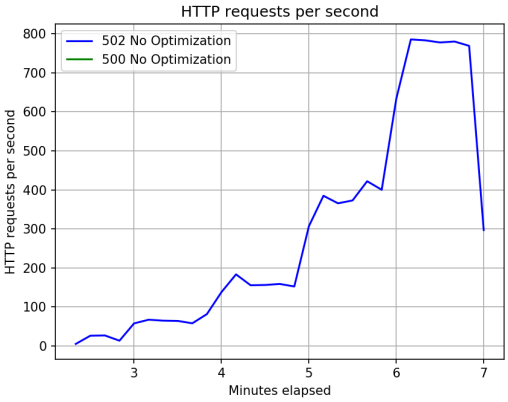


Figure 36: Workflow 5: Seller Adds An Item

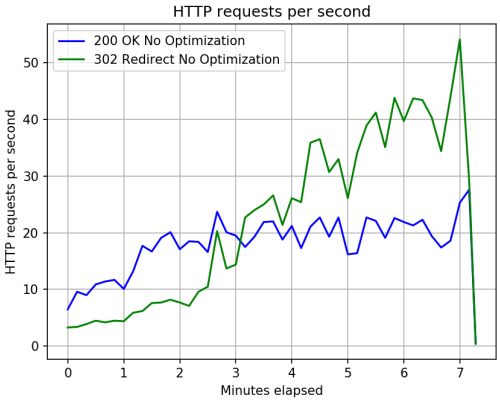
This workflow simulates the following scenario:

1. Seller signs in
2. Seller checks his profile
3. Seller adds an item
4. Seller checks the home page to verify if the item is present or not
5. Seller visits his profile

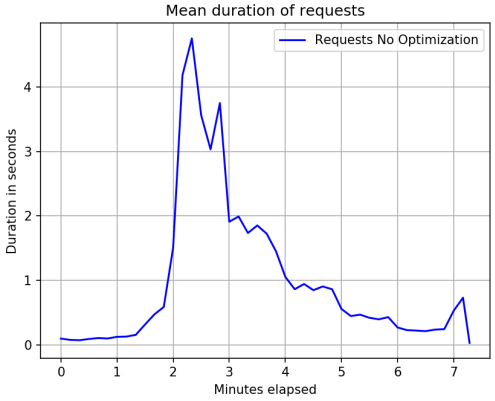
Bottleneck Analysis Initial load test without any optimization shows that there are major bottlenecks in the home page. We can see that 502 errors start immediately after 15 seconds. Also, the 200 and 302 responses per second are also very low (40). We can see that mean duration is also very high (4.5 seconds) which suggests that there are lot of improvements needed (especially home page).



(a) 5XX Responses



(b) 200, 300 Responses

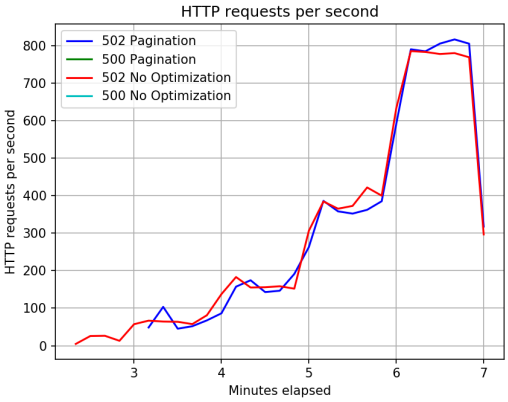


(c) Mean Request Duration

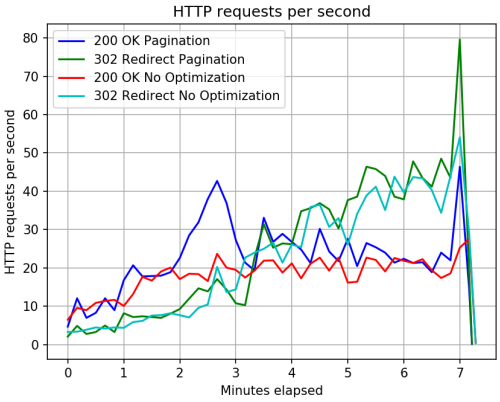
Figure 37: Initial Status Without Any Optimizations

Optimizations Below are different optimization techniques we tried and could improve the test workflow.

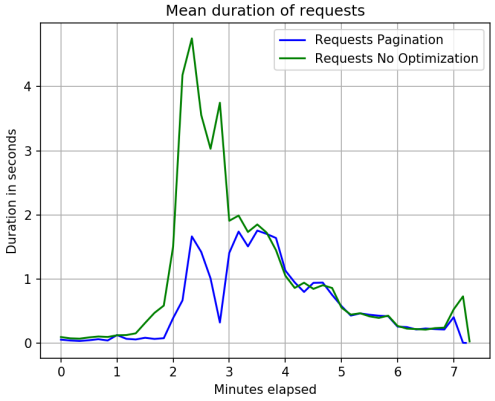
Pagination The home page shows all the items from the database. As we are seeding with 10,000 items, we're loading 10,000 items from the database whenever the home page is loaded. With pagination, we cut the number of items per page to 10 and from the below graph we can see that the mean response time is reduced to less than 2 seconds. The HTTP requests per second increased to 45 from 20 in phase 3. Also, the 502 responses started from phase 3 instead of starting from phase 1.



(a) 5XX Responses



(b) 200, 300 Responses



(c) Mean Request Duration

Figure 38: Workflow 5 After Pagination

Caching The next performance optimization we did is Caching. While pagination reduced the mean request time, adding Caching on top of Pagination surprisingly increased the mean request time a little. One reason might be frequent cache invalidation which means data needs to be loaded from the database freshly.

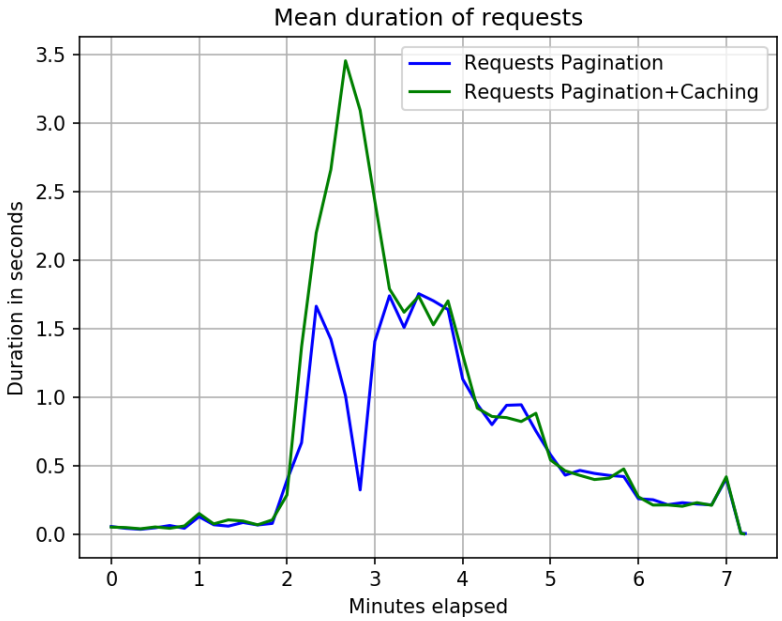


Figure 39: Mean Duration after Caching and Pagination

Indexing While we added indexing on top of Caching, it really didn't improve the performance much. One reason might be that as we did pagination and already cut down the items to 10 per page, indexing didn't have much scope to improve any performance (i.e., improvement due to indexing is not visible because we're fetching only 10 items every time).

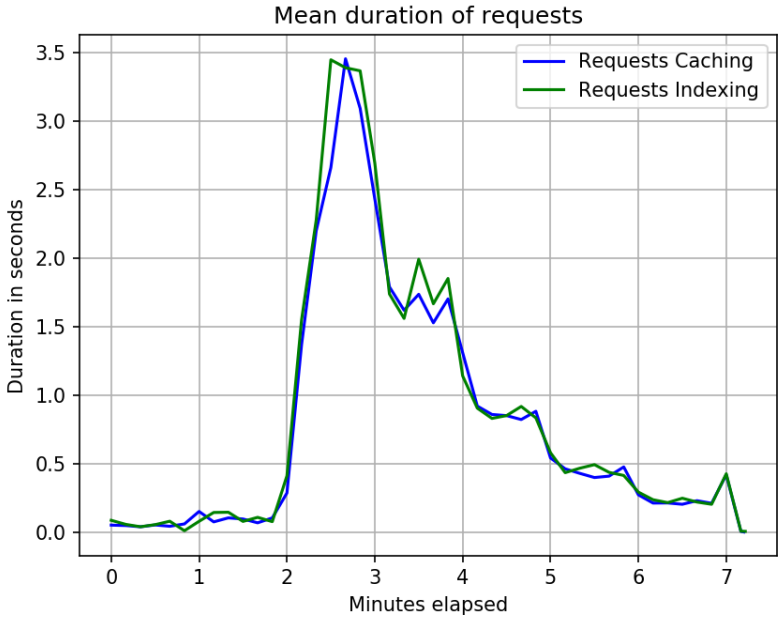


Figure 40: Mean Duration after Indexing

6 Scaling

After optimizing each individual workflow, we combined all our workflows in a single Tsung test framework under multiple sessions. We assigned different probabilities to every workflow based

on general reasoning and typical user behavior to assess the overall real-time performance of our application. The goal was to stress test the application end-to-end under different simulations to get a holistic performance metric. All the scaling tests use the same test scenario described below. Except for User Arrival Rate, scaling every other subsection uses the same arrival phases in Tsung flows.

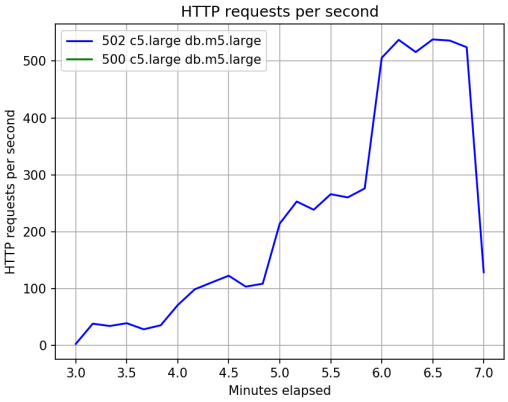
Test Scenario We used the following probabilities for each workflow.

1. **Workflow 1**: 35%
2. **Workflow 2**: 25%
3. **Workflow 3**: 20%
4. **Workflow 4**: 15%
5. **Workflow 5**: 5%

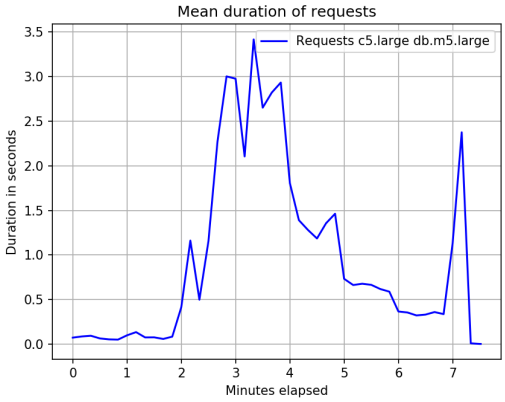
Tsung Arrival Phases Each phase is for 60 seconds.

1. **Phase 1**: 2 users/sec
2. **Phase 2**: 4 users/sec
3. **Phase 3**: 8 users/sec
4. **Phase 4**: 16 users/sec
5. **Phase 5**: 32 users/sec
6. **Phase 6**: 64 users/sec
7. **Phase 7**: 128 users/sec

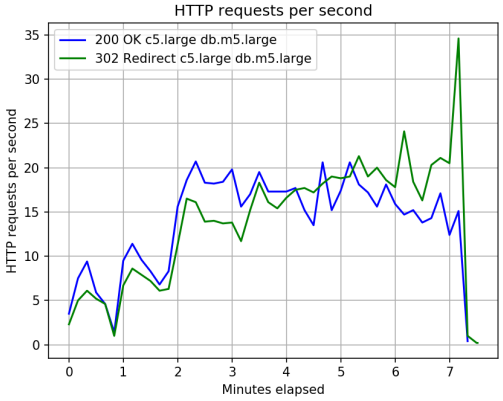
Before any kind of scaling, the performance of our website on a c5.large app instance and db.m5.large database instance showed a lot of 5XX errors and the mean request duration was more than 3 seconds. But as the Workflow-5 has lower probability, the number of 5XX errors are lower than that workflow.



(a) 5XX Responses



(b) Mean Request Duration



(c) 2XX Responses

Figure 41: Status of Website Before Scaling

6.1 Vertical Scaling

Scaling to app c5.large, database db.m5.xlarge We scaled the database from db.m5.large to db.m5.xlarge to evaluate if improvement in database would provide any optimizations. But the number of 5XX, 200, and 302 responses remained the same. We only found a little improvement in request mean duration.

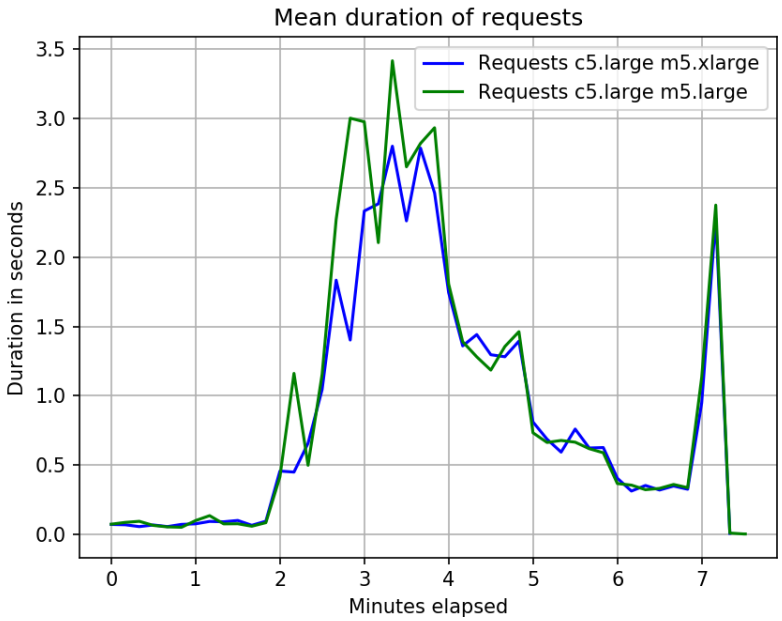
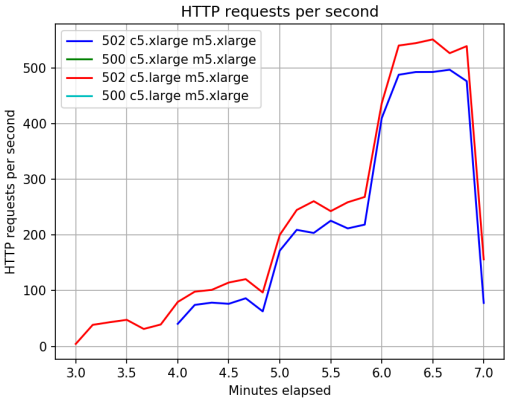


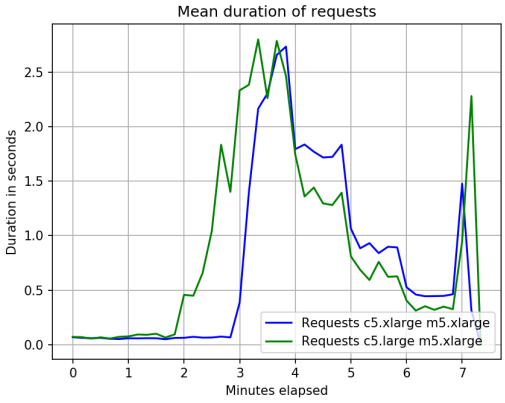
Figure 42: Decrease in Request Mean Duration After Scaling to db.m5.xlarge

Scaling our application to db.m5.xlarge proved that database was not a bottleneck.

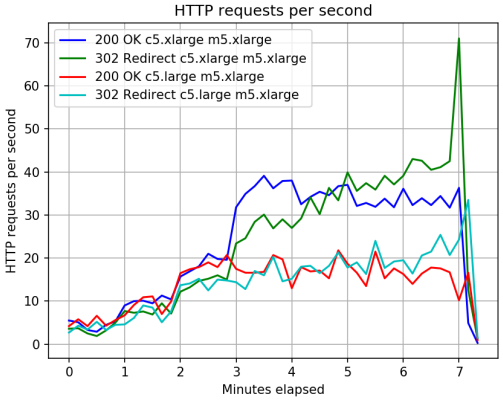
Scaling to app c5.xlarge , database db.m5.xlarge We scaled our app server from c5.large to c5.xlarge and kept the database at db.m5.xlarge. We saw decrease in 5XX responses, increase in 200 and 302 responses, and one additional phase improved in terms of request mean duration.



(a) 5XX Responses



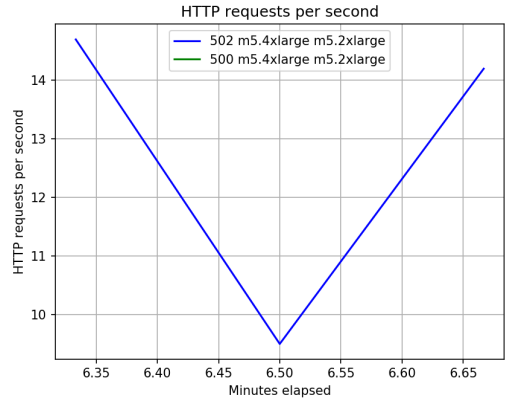
(b) Mean Request Duration



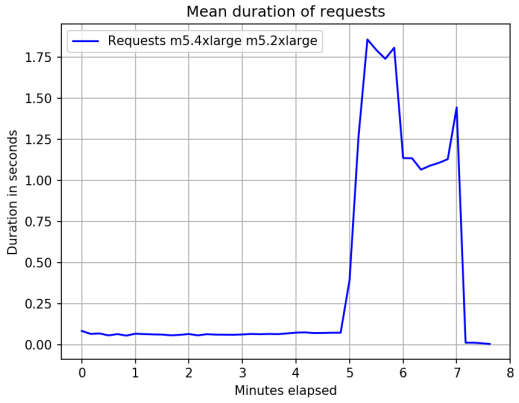
(c) 2XX Responses

Figure 43: Status of Website with app c5.xlarge database m5.xlarge

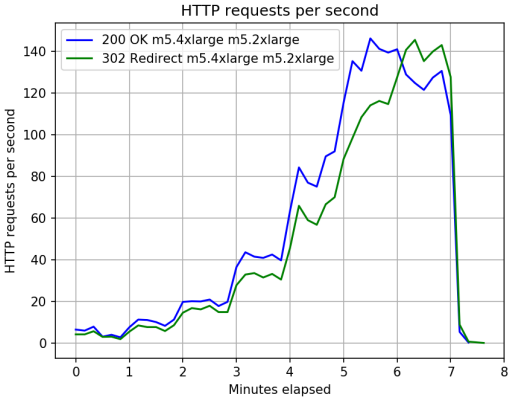
Scaling to app m5.4xlarge database m5.2xlarge We scaled our app server from c5.xlarge to m5.4xlarge and scaled the database from m5.xlarge to db.m5.2xlarge. We saw an appreciable decrease in 5XX responses, increase in 200 and 302 responses, and request mean duration stayed constant for 5 phases. But we noticed errors with connections being dropped during the 5th phase. At this point, we decided to scale horizontally to see if adding more servers will decrease the errors.



(a) 5XX Responses



(b) Mean Request Duration



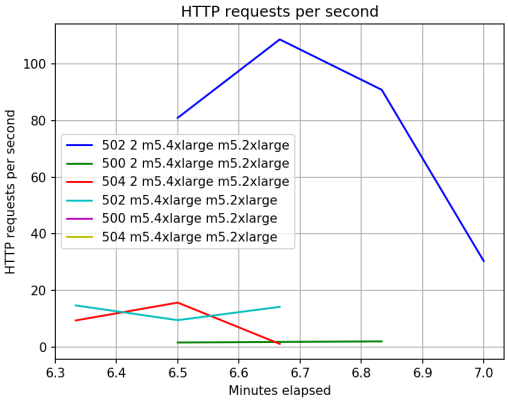
(c) 2XX Responses

(d) Errors

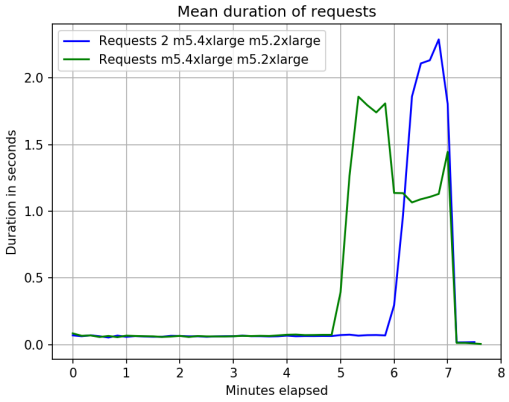
Figure 44: Status of Website with app m5.4xlarge database m5.2xlarge

6.2 Horizontal Scaling

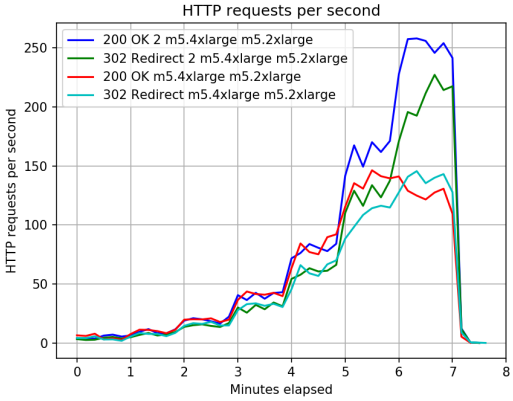
Scaling to 2 instances of app m5.4xlarge database m5.2xlarge After scaling horizontally to 2 instances of application server, we saw that there are no errors. We saw great improvements in 200 and 302 responses and 6th phase was being handled well. But in the last phase, with 128 users/sec we observed all kinds of 5XX responses. So, we decided to scale by a factor of 2 again.



(a) 5XX Responses



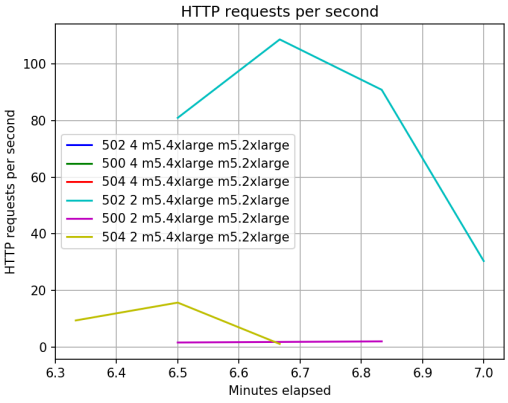
(b) Mean Request Duration



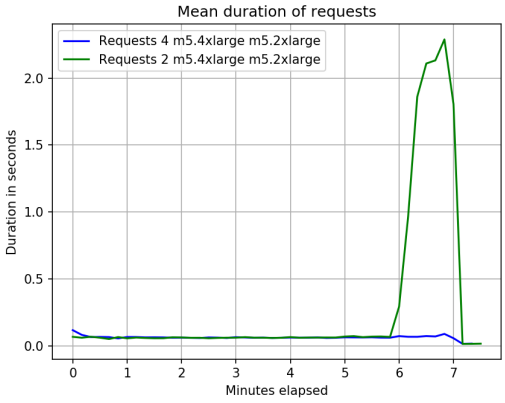
(c) 2XX Responses

Figure 45: Status of Website with 2 instances of app m5.4xlarge database m5.2xlarge

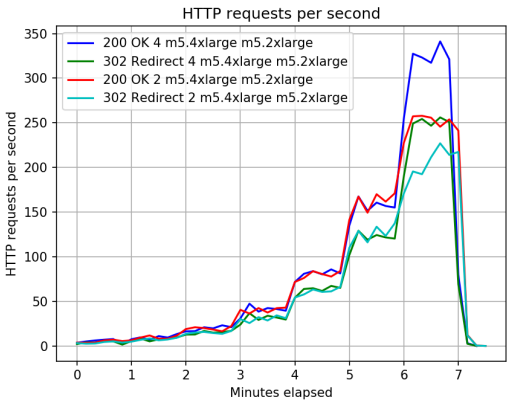
Scaling to 4 instances of app m5.4xlarge database m5.2xlarge After scaling horizontally by 2 we see that our website is very much optimized. We see no 5XX responses or errors with connections. All responses were 200 or 302. And all the phases have constant mean request duration which is less than 200 milliseconds.



(a) 5XX Responses



(b) Mean Request Duration

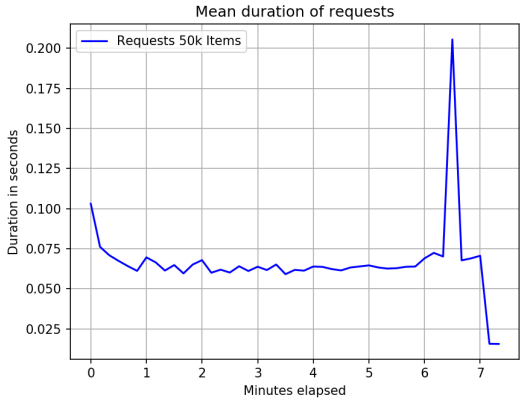


(c) 2XX Responses

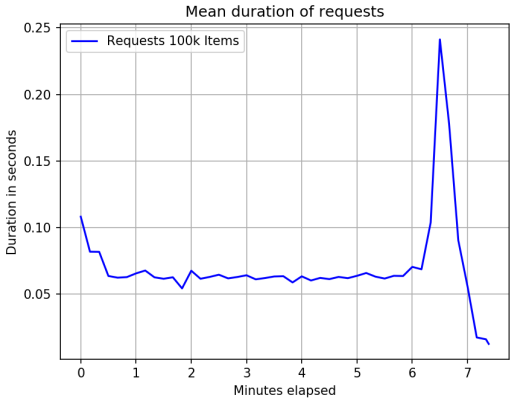
Figure 46: Status of Website with 4 instances of app m5.4xlarge database m5.2xlarge

6.3 Scaling Database Records

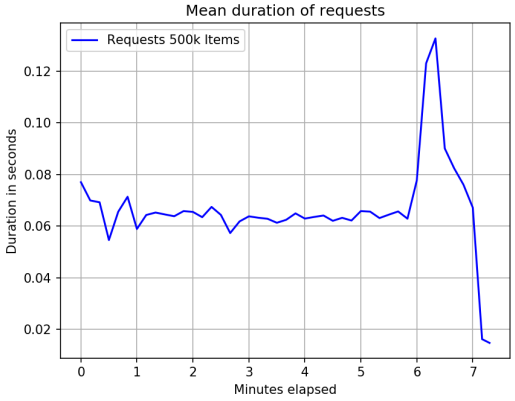
We started our load testing with 10,000 Items for 100 Sellers, 500 Orders and 500 Carts for each Buyer. We later scaled the number of database records to check if increase in these records will affect the website's performance. We scaled the records in our database and checked the performance of our website accordingly. We took the number of Items as reference and scaled other tables accordingly and did Tsung load testing on our website. We scaled from 10,000 Items to 50,000 Items, then to 100,000 Items and finally 500,000 Items! We observed no decrease in the performance of our website. There were no 5XX responses. Every response was 200 or 302 and all the users finished their requests. The request mean duration stayed below 200 milliseconds. We account some mishaps in request mean duration to the initial warming of load-balancers. This proves that database alone is not a bottleneck in our website.



(a) Request Mean Duration for 50k Items



(b) Request Mean Duration for 100k Items



(c) Request Mean Duration for 500k Items

Figure 47: Scaling DB Records

6.4 Scaling User Arrival Rates

In all the Tsung tests, we fixed the number of phases to 7 where the maximum arrival rate of the users is 128 users/sec. We tried to push this limit further and checked how our website performs with increasing arrival rate of users. We increased the number of users per second by factor of two in each phase and went up to 1024 users/sec. The results were quite astonishing and we can use this as a reference to handle the incoming traffic. We can also use this as a reference to consider the cost of handling users as AWS provides us the cost per hour for using a certain configuration of machines. We used AWS EC2 On-Demand Pricing as a reference for calculating the cost of these machines.

Criteria for cost/hr The criteria for evaluating the cost/hr incurred for every app and database configuration at different user arrival rates was the fact that there are no 5XX errors generated and the mean request duration is below 200 milliseconds.

User Arrival Rate	App Configuration	Database Configuration	Cost/hr
2	t3.micro	t3.micro	0.0208
4	c5.large	m5.large	0.181
8	c5.xlarge	m5.xlarge	0.277
16	m5.2xlarge	m5.2xlarge	0.768
32	m5.4xlarge	m5.2xlarge	1.152
64	2 m5.4xlarge	m5.2xlarge	1.92
128	4 m5.4xlarge	m5.2xlarge	3.456
256	4 m5.16xlarge	m5.4xlarge	13.056
512	8 r6g.16xlarge	r6g.4xlarge	26.61
1024	16 r5.24xlarge	r5.8xlarge	98.784

Table 1: Cost/hr incurred for various configurations

Plotting the results of above table shows us the diminishing returns in terms of scaling. We could not plot the graph in linear scale as higher numbers like 512 and 1024 dominate the real-estate of the graph. So, we used a **semi-log** graph on the x-axis and excluded 1024 users/sec result to get a good-looking graph.

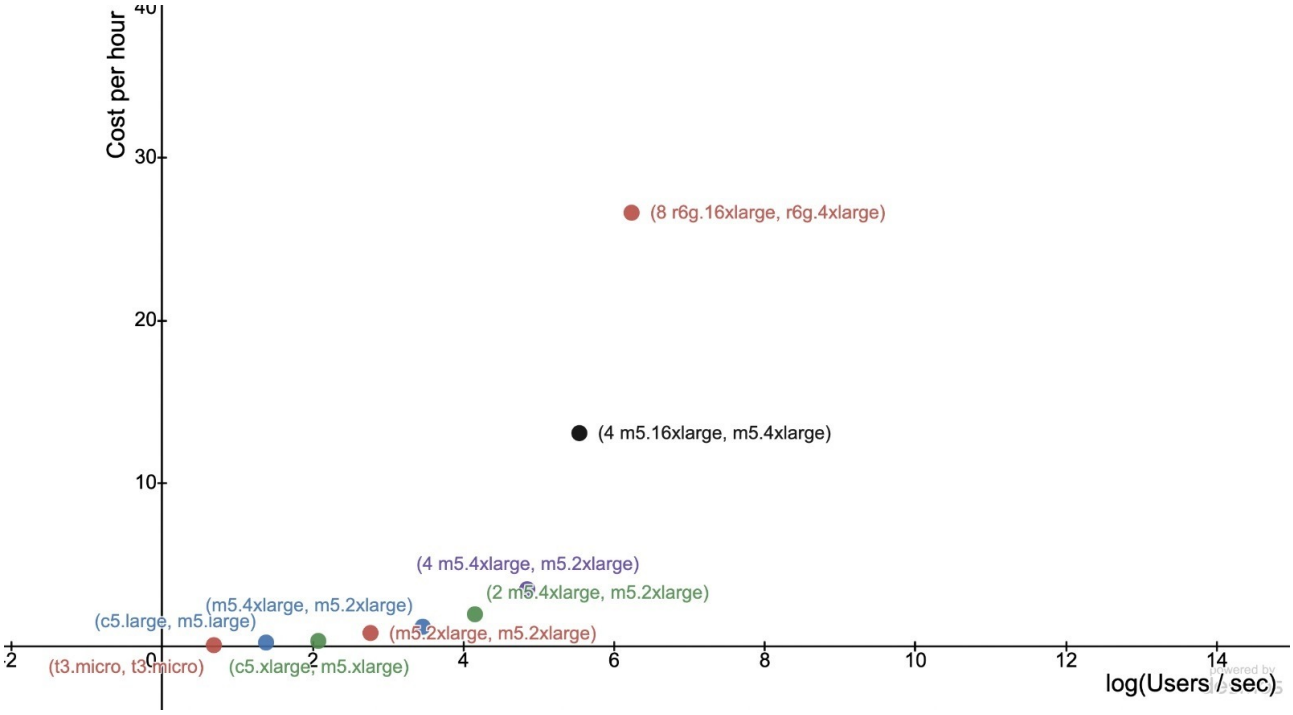


Figure 48: Cost to Maintain a User Arrival Rate

6.5 Solutions to Diminishing Returns

The diminishing returns we see in the graph 48 can be due to any number of reasons starting from AWS's inner-workings to our website's design and implementations. Nevertheless, we need to grow and address the rising demand from the user's front. Some possible solutions for this include -

- We can introduce multiple load-balancers and distribute the load among them as having a single load balancer might cause issues.
- Using network load balancer instead of application load balancer can also help in handling high numbers of concurrent users.
- Switching to Golang might also give us improvements. Golang has gained popularity for server-side applications. Golang is a faster programming language as compared to Ruby because Golang doesn't need to be interpreted. Golang is a statically typed language while Ruby is a dynamically typed language. The cost of creating a Goroutine is cheap and we can have thousands of Goroutines running concurrently for handling the requests.
- Splitting up our infrastructure in multiple regions can help in reducing load at each region, which could help in handling requests faster for those who are closer to the region.

7 Future Work

As mentioned in the prior feature section, the eKirana application provides minimal essential features. The list of features built and developed are reasonable enough to achieve and essential for any modern fully-functional e-commerce website that is in production.

1. The ability to add multiple addresses for a Buyer will provide the flexibility to order an item to a residential address or say the office address. It will also allow buyers to gift items to each other.
2. A payment gateway to pay for items is at the core of any e-commerce website. With all the payment solutions available these days, it is not too difficult to integrate payment APIs within this application.
3. A discount model that keeps track of discounts on a item would be easier for sellers to announce price reductions rather than editing an item every time.
4. Images are a huge part of any user friendly application. We omitted using any images as they are static assets which will not effect load testing. But addition of a single image to an item will make a huge difference in terms of incoming traffic.
5. Search Engine that supports item names, types and comments would also be a good feature to have. This adds a whole new dimension to back-end analytic and load testing.

Apart from this there are a plethora of other features that could be integrated. We have not taken care of any web security which is an integral part of any modern internet service.

8 Conclusion

The following report provides the motivation for an e-commerce website like eKirana along with the features and services offered. We also provide all the implementation and abstraction details inculcated. We then stress tested the application via different workflows to simulate real-user behaviours to see where our application breaks or performs poorly. We finally do a combined load test of the website using Tsung by assigning various probabilities to each workflow. We discussed how optimization techniques such as pagination, indexing, n+1 query optimization and extensive caching affect the performance of the website across different workflows. The optimizations are evident through the various graphs embedded. We then proceeded to scale our

website vertically and horizontally to find the right set of application and database combinations for different user arrival rates. We also scaled the database records to make sure database is not a bottleneck in our application. However, a powerful server with a weak database will also result into many issues. So, while scaling, we also scaled our database instances to get scalable performance.

We then exponentially increased the user arrival rate and checked the performance of our system and tried to find the right configuration to deal with certain user arrival rates. Thus, we were able to see that throwing money at the problem will not always work and argued a few potential solutions. We also provided future improvements and features for our project so as to create a more user friendly experience for our customers using such a platform.