PriceKing – Real Time Best Deal Search and Analytics

A Project Report

Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By

Deven Pawar

Hardik Joshi

Naiya Shah

Vinay Bhore

May 6, 2014

APPROVED FOR COMPUTER ENGINEERING DEPARTMENT

Chandrasekar Vuppalapati

Digitally signed by Chandrasekar Vuppalapati
DN: cn=Chandrasekar Vuppalapati, o=San Jose State University, ou=Computer Engineering, email=chandrasekar.vuppalapati@sjsu.edu, c=US
Date: 2015.05.01 00:57:11 -07'00'

Chandrasekar Vuppalapati, Technical Advisor

Dr. Dan Harkey, Academic Advisor

ABSTRACT

PriceKing – Real Time Best Deal Search and Analytics

By,
Deven Pawar,
Hardik Joshi,
Naiya Shah,
Vinay Bhore

The e-commerce vendors and giants such as Amazon, eBay, Target, Walmart and many more today have captured the world of commerce; where finding a right deal for a desired product is challenging! Either you have to be really lucky or scavenge through all available e-commerce web sites to get a good deal. Although there are many individual vendors who claim that they get you best deal however the user can't rely on a single vendor claiming the best deal. In this high tech world, the things are moving from desktops, laptops to the mobiles and tablets to access things, where we realized that this problem is not addressed on mobile platforms. There are some unknown small businesses, which get affected due to the problem mentioned above that needs to be addressed to build trusted win-win commerce ecosystem where merchants win by selling their products and customers win by buying at cheapest price available.

This project uses distributed stream processing and analytics tools which will collect the continuous stream of big data from many mobile users, distributes their product search requests and processes them in parallel to get the best deals for the specific product item. Furthermore, it crunches the searched data from various vendors and filters it based on the criteria specified by the customer. The goal is to create a user centric system with user engagement analytics. Another motivation behind the project is to monitor user-shopping patterns and search history to provide recommendations through notifications. We also plan to create a web portal for small businesses, where they can post their deals and offers, which will be consumed by our services to crunch and provide cheapest deals for our end users.

To fulfill the problem statement, we propose PriceKing a mobile and tablet application hosted on cloud, which will search for the cheapest deals from many vendors, small businesses with the coupons and offers for the found deal on a mobile and tablet podium. PriceKing enables user to search products using various media such as text, speech or image recognition systems. PriceKing will also provide a web application that helps small businesses to promote their business by advertising their products and offers to the mobile end user. Small businesses can also track their product search results by providing graphical analysis on their profile.

Table of Contents

List of Figures

List of tables

1. Introduction

1.1 Project goals and objectives

The goal of this project is to create a mobile application that will provide best deals for any searched product. The system uses distributed stream processing and analytics tools, which will collect the continuous stream of big data from many mobile users, distributes their product search requests and processes them in parallel to get the best deals for the specific product item. Furthermore, it crunches the searched data from various vendors and filters it based on the criteria specified by the customer. The goal is to create a user centric system with user engagement analytics. Another motivation behind the project is to monitor user-shopping patterns and search history to provide recommendations through notifications. We also plan to create a web portal for small businesses, where they can post their deals and offers, which will be consumed by our services to crunch and provide cheapest deals for our end users.

1.2 Problem and motivation

E-commerce market is growing rapidly due to mobile and tablets usage. People spend more time on mobile and tablets than PC. In 2016 approximately 131.4 million users will have made at least one purchase on their mobile devices via browser or app. [1]Many individual price comparison websites claim that they provide cheapest product deal and there are vendors, which provides digital coupons that are unused most of the time. Of the 305 billion CPG (Consumer Product Goods) coupons distributed throughout the country, U.S. consumers only redeemed 2.9 billion of those CPG coupons. [2] Users find it difficult to go through no of websites for searching and comparing deal and then to find valid coupons available for that product. Thus, in this large, fast and highly competitive market there are some issues, which need to be addressed. Single unified search platform

that performs price comparison and search coupons that can be applied to the product will save lot of time and maximize coupon utilization. Also, for small businesses it's very difficult to target such a large market with limited resources. Dashboard with user engagement analytics, user search patterns, user reviews and ratings and other analytics will be useful for them to expand their business and target large no of users.

1.3 Project application and impact

Our aim is to maximize the use of coupons and provide unified search platform that find cheapest deal across all the major merchants. Also, we want to encourage small businesses by providing dashboard to take the advantage of e-commerce market and be beneficial via listing their products and coupons for free. Thus, small businesses can expand and target large audience of mobile and tablet users. Also, useful analytics and users' insight will provide them magnified view of user's behavior. Many vendors provide student discount coupons which are either printed or only available on website of merchants. We have the way to merge it and embed this functionality in our application that will help students to get best offers. Also, by embedding coupons into our application we want to cut down paper coupons usage. Digital coupons are eco friendly and always easy to carry.

1.4 Project results and expected deliverables

Our project is divided into two different components, android application and customer dashboard. Android application is end user product that will provide unified search for price comparison engine and coupons with additional features like voice search, image search, barcode search, geo-notification, save deals etc. Customer dashboard is oriented to small business user. Small business users will have platform to post products and coupons, manage existing listing and magnified view of users insight via user engagement analytics and user activities. Also, for managing small business user, admin will have

control over small business user's profile via reports like most active users, most selling products, most selling coupons etc.

Deliverables for PriceKing includes the following things-

1) Project Abstract
2) Working Android mobile application deployed on google play store
3) Cloud hosted web application for small business users
4) Storm Kafka architecture deployed on cloud
5) Source code of both the application
6) Project Report

Project Abstract – Gives an overview of the application along with the motivation factors, ideas, problems addressed and project features.

Working Android mobile application deployed on google play store –

Android app is targeted for the end users who are searching for a deal. So android app is the crucial deliverable. It is deployed on the Google Play Store so the users can download it and start using on their android phones.

Cloud Hosted Web app for small business users – For small business users there is a web application through which the vendors get a platform to sell and advertise their app. Vendors are also provided with analytics to expand/improve their business. Admin has insight to all the vendor activities.

Storm Kafka Architecture deployed on cloud – The architecture of the PriceKing app is also deployed on cloud.

Source code of both the application – Source code of both the application is there on github.

Project Report – Project report includes the detailed instruction of the application usage and benefits. It includes abstract, market values, application algorithm, snapshots, implementation, testing summary, research.

1.5 Market Research

Economical and Social Aspects:

This way Manufacturers earn more profit and end customers or buyers get it at a cheaper price making it a win-win situation for both manufacturers and buyers. The problem comes in when there is multiple ecommerce websites selling same stuff with different add on services and different prices, which confuses buyers, which site to buy from?

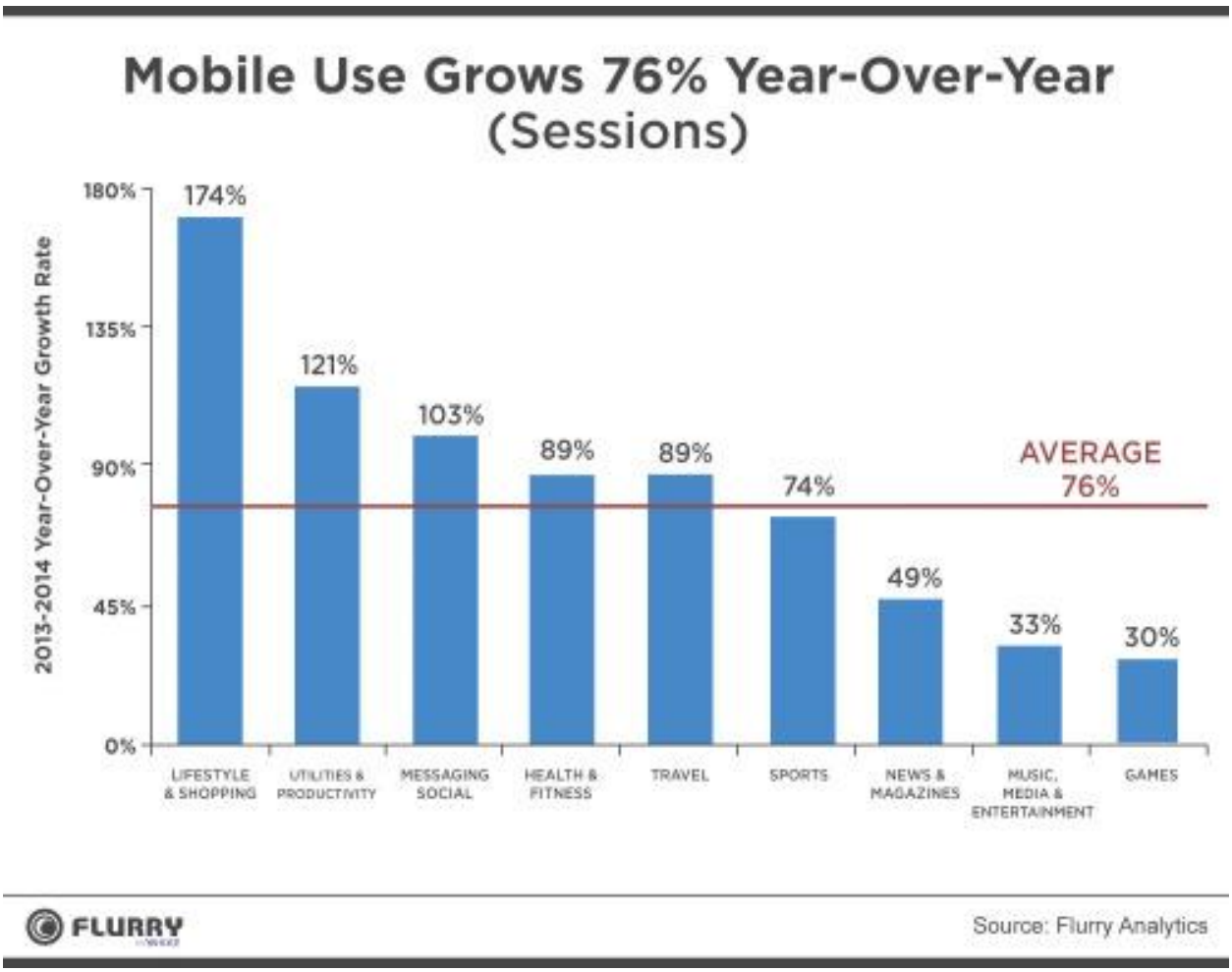
Our application deals with some of the economic and social aspects dealing with the problem mentioned above.

- Cost Comparison: The application crawls through many merchant websites to find the cheapest price of the item buyer is looking for. The application also considers the promotions offered by the merchant or any third party on the searched item making it really affordable for the buyer.
- Merchant advertisement: Not only the buyers but also manufacturers are benefited by our application. The application finds the cheapest deal and navigates the buyer on merchants website to proceed with purchase where the merchant can also practice customer engagement strategies. With this strategy our application targets small businesses that have a lot better deals but don't have enough budget for advertisement.
- Customer Interest and buying pattern: Every business has a keen focus on customer's likes and dislikes. Many large-scale vendors and commerce giants spend millions on doing this analysis. Our application provides a dashboard providing customer's buying habits and his interests for free for every vendor registered with us that helps businesses provide custom deals for an individual user.
- Revenue on referral: Our application generates revenue on navigating every buyer on merchants website. Since our application involves small business as well, it generates small amount of revenue by advertising their products through our application. Thus, advertisement is also one of the important sources of revenue in our product.

- Time: We know that every minute of our customer is important in todays fast moving life. This is the reason why we offer various cheap deals from different merchants under one roof through our product. This saves a lot of time, as the user doesn't have to scavenge through different merchants and manually search for the cheapest product.
- Customer Impact: Customers are benefited by getting the cheapest deals on product search and also get the alerts for the newer deals on their earlier searches. This allows our application to understand its user's buying pattern, which makes our product more intelligent to offer you the deals of your interest in future. The product takes different types of input from the user (text, image, voice) according to the users convenience.
- Business Impact: Businesses get more referrals and more customers visiting their website. Small businesses can save huge amount of printing cost of advertising their products as well as coupon pamphlets by posting them on our application. Moreover the business can be analyzed and improved based on different geographical statistics that would be displayed through our product, which would help in future decision-making process.

Environmental Impacts:

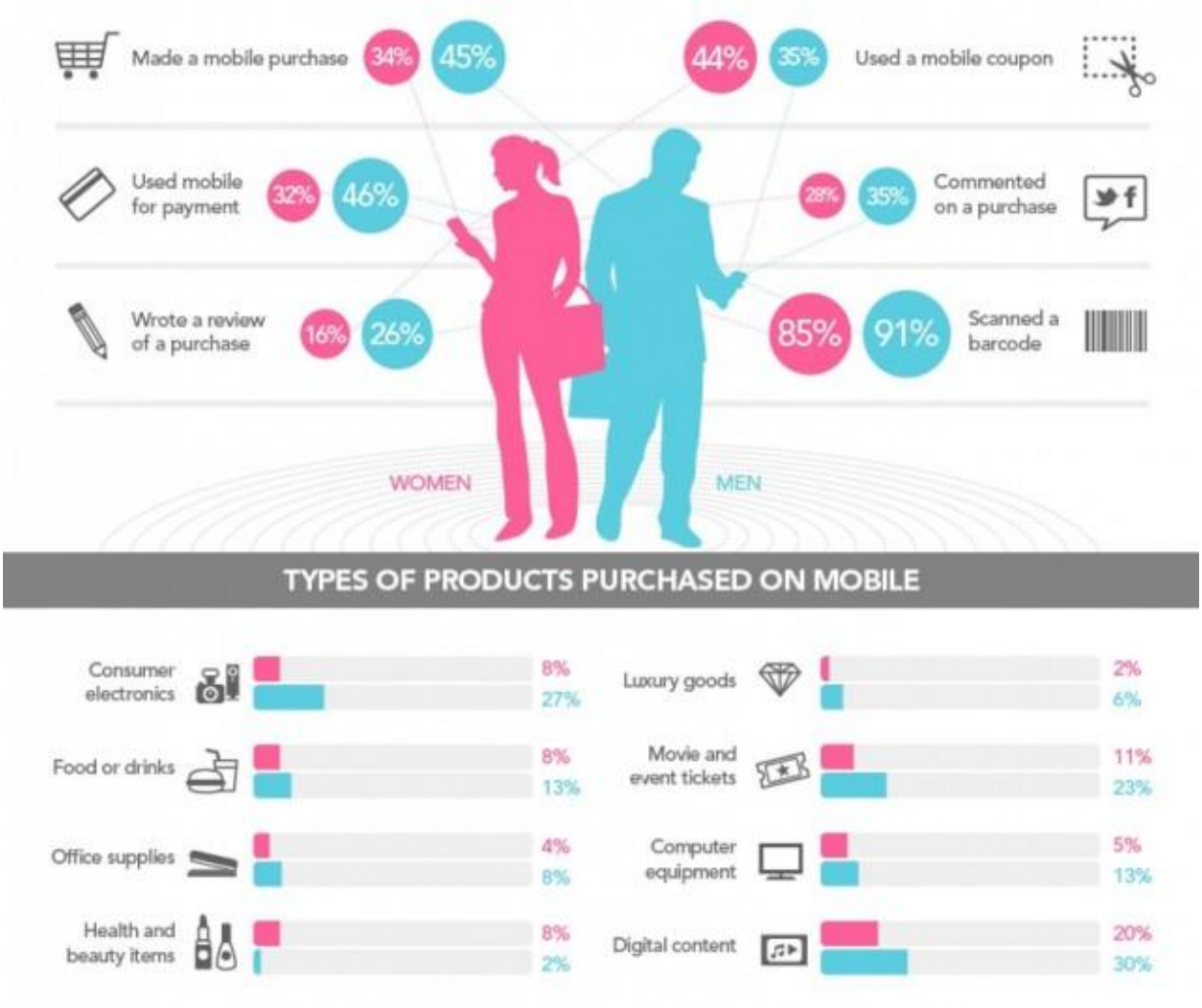
- No paper usage: As we encourage small businesses to post their coupons on our dashboard rather than printing it and distributing it, we can email these coupons to subscribed users which will save paper. We also encourage small businesses to validate coupons via application or email instead of asking users to print the coupon.
- Energy Savings: You don't have to always plug in your smart phone like other consumer electronics and energy consumed by mobile devices and tablets is lesser than other consumer electronics, which will result in energy savings.



Priceking Business Model - 1

Ecommerce was the real mobile star of the past year. According to mobile analytics firm Flurry, shopping app usage grew faster than any other category of apps. Sessions on shopping apps on iOS and Android devices increased by 174% year-over-year, including 220% on Android alone. That's up from 77% last year.

Utilities and productivity apps were the second-fastest growth category, up 121%, while messaging and social took third place, with 103% growth. Overall app use was up 76%, so all three app categories outpaced the total marketplace



Priceking Business Model - 2

1.6 Project report Structure

The project report is structured into eight main sections. The first section is Introduction, which gives brief overview of problem statement, motivation and needs, problem application along with Economical and Environmental impacts, project result and deliverables and market research.

The second chapter includes project background and Related Work. These consist of background along with the used technologies, State-of-the-art technologies and Literature Survey of the project.

Third section elaborates project plan and schedule. This starts with project team and tasks followed by project milestone schedule and detailed schedule.

Chapter four covers software system requirements such as domain and business requirements, customer oriented requirements, functional, non-functional and behavior requirements, context and interface requirements followed by technology and resource requirements.

The chapter five focuses on preliminary software system design. This covers different types of design required for the project such as system architecture design, database design, user interface design, component API and logic design.

Tools and adopted standards are covered in chapter six whereas chapter seven concentrates on testing and experiment plan.

Then the future prospects of the project are discussed. Like what can be added to make the application better.Finally, the conclusion of the project.

2. Background and Related Work

2.1 Background and used technologies

API and Application Layer	• Restful Web Service • Spring MVC • Tomcat
Database Layer	• MongoDB
UI Layer	• HTML5 • CSS3 • JQuery • JavaScript • JSP • Twitter Bootstrap • AngularJS
Messaging	• Apache Kafka • Storm • Esper
Deployment	• Play Store • Amazon EC2
Mobile Client	• Android
Testing	• Selenium Web Driver • Testdroid • Rest Console • JUnit

Table 2.1 Background and used technologies

API and Application layer

Application is developed in Spring MVC framework as it is the best solution for management of web application page flow. Restful web-services are used for client-server communication through HTTP. Apache tomcat server as it provides HTTP web-server environment for java code.

RESTFul
Web Services

Database Layer

MongoDB is a cross-platform document oriented database.

User Interface

For building responsive UI, Bootstrap is used along with AngularJS to reduce ajax calls. HTML5, jQuery and CSS3 for making the application attractive. Responsive UI means the user interface or the front end is so flexible that it can adjust to any screen.

Middleware Architecture

Apache kafka is a message queue rethought as a distributed commit log. It follows publish-subscribe messaging style, with speed and durability built in. Kafka uses zookeeper. Storm runs continuously consuming the data from spouts and passes the data down the pipeline-bolts. Combined spouts and bolts form a topology and a topology is written in java for building the architecture for PriceKing

Deployment and versioning

GitHub is used for our version control as it is a good source for team collaboration. Web application is deployed on AWS and android mobile app is on google play store.

Testing

Selenium is easy to use and perform automated functionality testing. Rest Console for API testing, Testroid for Mobile app testing and JUnit for functional testing

Agile Project Management

Team Foundation Server mainly for project management to maintain our user stories, epics, use cases and the according sprints followed.

2.2 State-of-the-art technologies

There are certain websites, which displays the cheapest deal from various APIs. PriceKing is the web as well as mobile app, which not only searches for cheapest deal but also associates the best discount coupon available for that particular search. Moreover, small business vendors will get a platform to sell their product, which is the win-win situation for customers as well as merchants. Small business merchants will also be provided with the analytics that will prove very useful to expand and popularize their business.

2.3 Literature survey

Price comparison website is a vertical search engine that shoppers use to filter and compare products based on price, features, and other criteria. Most shopping sites aggregate product listing from many different retailers but do not directly sell product themselves. There are many criteria involved in price comparison but we found most three common techniques as below. [5]

- Price comparison sites can collect data directly from merchants. Retailers who want to list their products on the website then supply their own lists of products and prices, and these are matched against the original database. A mixture of information extraction, fuzzy logic and human labor does this.

- Comparison sites can also collect data through a data feed file. Merchants provide information electronically in a set format. This data is then imported by the comparison website. Some third party businesses are providing consolidation of data feeds so that comparison sites do not have to import from many different merchants. This enables price comparison sites to monetize the products contained in the feeds by earning commissions on click through traffic.

- Yet another approach is to collect data is through crowdsourcing. This lets the price comparison engine collect data from almost any source without the complexities of

building a crawler or the logistics of setting up data feeds at the expense of lower coverage comprehensiveness. Sites that use this method rely on visitors contributing pricing data. Unlike discussion forums, which also collect visitor input, price comparison sites that use this method combine data with related inputs and add it to the main database though collaborative filtering, artificial intelligence, or human labor. Data contributors may be rewarded for the effort through prizes, cash, or other social incentives. Wishabi, a Canadian based price comparison site, is one example that employs this technique in addition to the others mentioned.

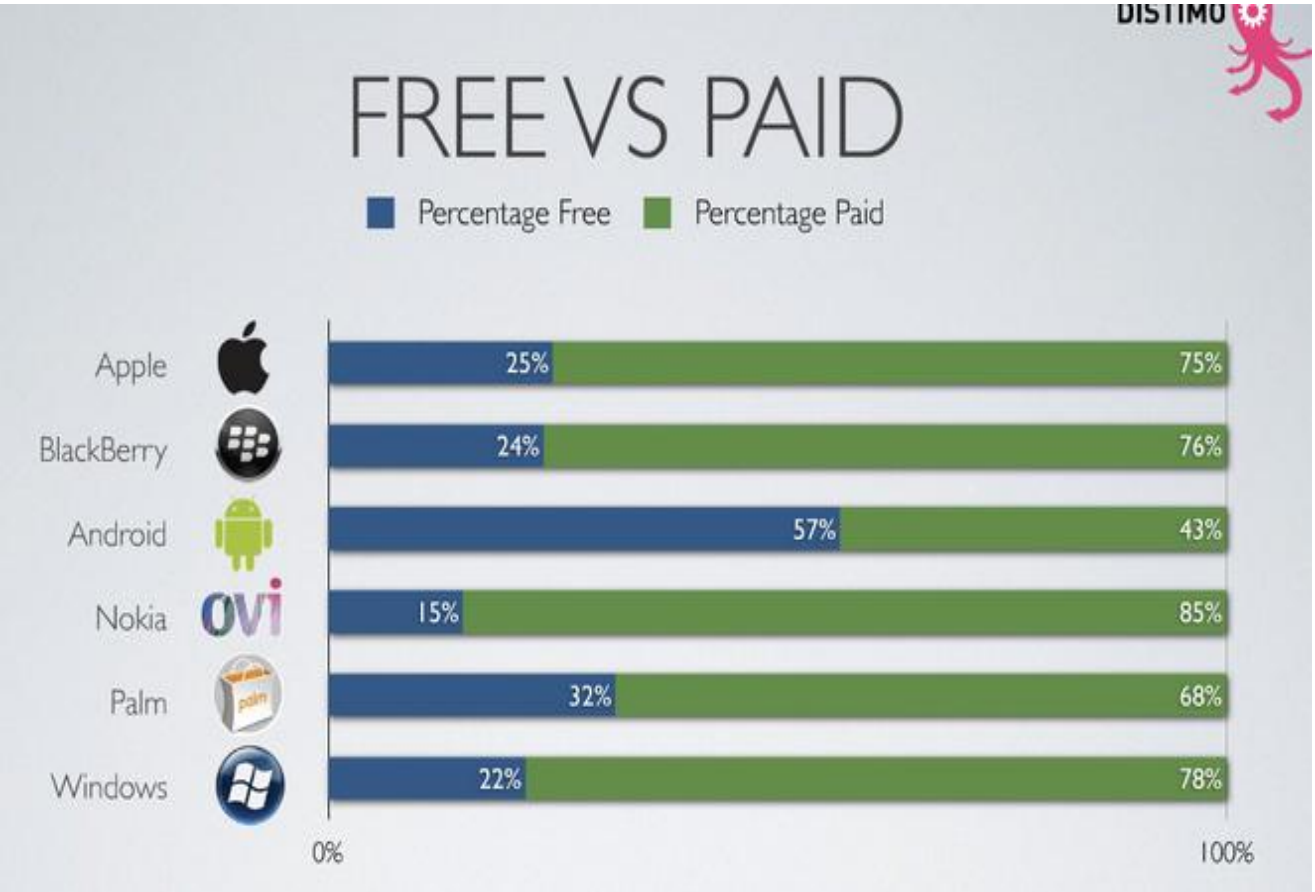
We also found price comparison site operate according to several business model among which most three common business models are as below.

- Users can access the comparison sites for free, while sellers have to pay a fee. This fee usually takes the form of cost-per-click and paid every time when consumer is referred to the seller's website from the comparison site
- Offering product and price comparison services for free to both sellers and buyers and relies on advertising as a source of revenue.
- A third, although less common model is to have consumers pay a membership fee to access the comparison site, while sellers are listed for free.

For our final project we are crawling web by hitting APIs provided by merchants and not depending on merchants to provide us data, as we want to make it as real time as possible. We are also filtering data very strictly to remove noise from search result so that consumer has most accurate result according to search query. [7] Many of the price comparison websites are not doing this as it provide result based on keywords which may not be related to listed product. We are adapting the most commonly used business model where consumer can access site for free and merchants have to pay a fee. In addition to this we want to keep it free for small business user and will only charge small amount of Fee if the consumer is buying product from small business user.

Why building android app is given priority over iOS app?

Android has several key advantages for the first-time entrepreneur. The android platform is based on open-source software, which makes the barrier-to-entry for cash-strapped developers guite low. This allows for you to use crowd-created plugins and frameworks which makes creation much simpler. For example, if you wish to create a mobile game for android instead of reinventing the wheel by building your own game framework you can use one like AndEngine. Because of Android's open source licensing, it has acquired a global user base. It has an 85% market share of the world's mobile devices. The main markets for android apps, the google play and kindle stores, also have fairly lenient restrictions on the types of the apps you can submit to them, allowing you to take advantage of the new technologies like crypto currencies when building your app. Android because it is faster and cheaper to develop apps for this platform, but if they want bigger revenue, they'd opt for iOS.



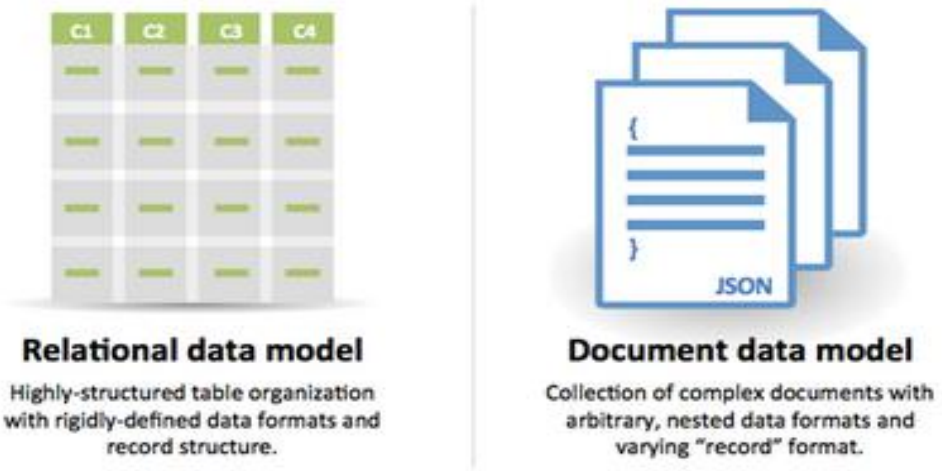
App Survey

Why NoSQL?

The idea that SQL and NoSQL are in direct opposition and competition with each other is flawed one, not in the least because many companies opt to use them concurrently. As with all the technologies, there really isn't a 'one system fit all' approach, choosing the right technology hinges on the use case. If your data needs are changing rapidly, you need high throughput to handle viral growth or your data is growing fast and you need to be able to scale out quickly and efficiently, may be NoSQL is not the right option. Bit if the data isn't changing in the structure and you are experiencing moderate manageable growth, you r needs may be best met by SQL technologies. Certainly, SQL is a good option as well for certain requirements.

	SQL	NoSQL
Data storage	Stored in a relational model, with rows and columns. Rows contain all of the information about one specific entry/entity, and columns are all the separate data points; for example, you might have a row about a specific car, in which the columns are 'Make', 'Model', 'Colour' and so on.	The term "NoSQL" encompasses a host of databases, each with different data storage models. The main ones are: document, graph, key-value and columnar. More on the distinctions between them below.
Schemas and Flexibility	Each record conforms to fixed schema, meaning the columns must be decided and locked before data entry and each row must contain data for each column. This can be amended, but it involves altering the whole database and going offline.	Schemas are dynamic. Information can be added on the fly, and each 'row' (or equivalent) doesn't have to contain data for each 'column'.
Scalability	Scaling is vertical. In essence, more data means a bigger server, which can get very expensive. It is possible to scale an RDBMS across multiple servers, but this is a difficult and time-consuming process.	Scaling is horizontal, meaning across servers. These multiple servers can be cheap commodity hardware or cloud instances, making it alot more cost-effective than vertical scaling. Many NoSQL technologies also distribute data across servers automatically.
ACID Compliancy (Atomicity, Consistency, Isolation, Durability)	The vast majority of relational databases are ACID compliant.	Varies between technologies, but many NoSQL solutions sacrifice ACID compliancy for performance and scalability

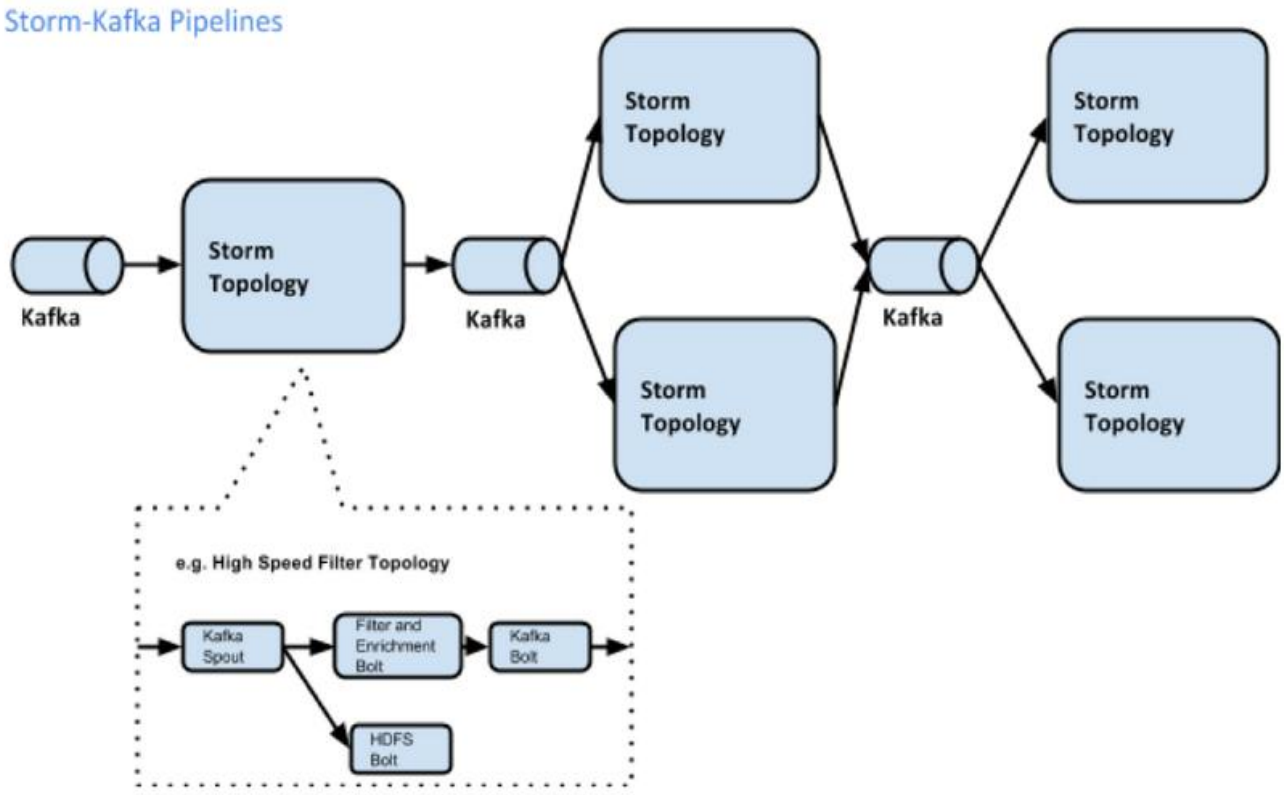
Table 2.2: SQL v/s NOSQL



Why to use Kafka and Storm?

Apache Kafka is a message queue rethought as a distributed commit log. It follows the publish-subscribe messaging style, with speed and durability built in. Kafka uses Zookeeper to share and save state between brokers. Each broker maintains a set of partitions: primary and/or secondary for each topic. A set of Kafka brokers working together will maintain a set of topics. Each topic has its partitions distributed over the participating Kafka brokers and the replication factor determines, intuitively, the number of times a partition is duplicated for fault tolerance. While many brokered message queue system have the broker maintain the state of its consumers, Kafka does not. This frees up resources for the broker to ingest data faster.

Storm has many use cases:real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to setup and operate.



Rest over SOAP web service implementation

REST is easier than SOAP -

I'm not sure what developers refer to when they argue that REST is easier than SOAP. Based on my experience, depending on the requirement, developing REST services can quickly become very complex just as any other SOA projects. What is your service abstracting from the client? What is the level of security required? Is your service a long running asynchronous process? And many other requirements will increase the level of complexity. Testability: apparently it easier to test RESTFul web services than their SOAP counter parts. This is only partially true; for simple REST services, developers only have to point their browser to the service endpoints and a result would be returned in the response. But what happens once you need to add the HTTP headers and passing of tokens, parameters validation… This is still testable but chances are you will require a plugin for your browser in order to test those features. If a plugin is required then the ease of testing is exactly the same as using SOAPUI for testing SOAP based services.

RESTFul Web Services serves JSON that is faster to parse than XML -

This so called "benefit" is related to consuming web services in a browser. RESTFul web services can also serve XML and any MIME type that you desire. This article is not focused on discussing JSON vs XML; and I wouldn't write any separate article on the topic. JSON relates to JavaScript and as JS is very closed to the web, as in providing interaction on the web with HTML and CSS, most developers automatically assumes that it also linked to interacting with RESTFul web services. If you didn't know before, I'm sure that you can guess that RESTFul web services are language agnostic.

Regarding the speed in processing the XML markup as opposed to JSON, a performance test conducted by David Lead, Lead Engineer at MarkLogic Inc, find out to be a myth.

REST is built for the Web -

Well this is true according to Roy Fielding dissertation; after all he is credited with the creation of REST style architecture. REST, unlike SOAP, uses the underlying technology for transport and communication between clients and servers. The architecture style is optimized for the *modern* web architecture. The web has outgrown is initial requirements and this can be seen through HTML5 and web sockets standardization. The web has become a platform on its own right, maybe *WebOS*. Some applications will require server-side state saving such as financial applications to e-commerce.

Caching -

When using REST over HTTP, it will utilize the features available in HTTP such as caching, security in terms of TLS and authentication. Architects know that dynamic resources should not be cached. Let's discuss this with an example; we have a RESTFul web service to serve us some stock quotes when provided with a stock ticker. Stock quotes changes per milliseconds, if we make a request for BARC (Barclays Bank), there is a chance that the quote that we have receive a minute ago would be different in two minutes. This shows that we cannot always use the caching features implemented in the protocol. HTTP Caching be useful in client requests of static content but if the caching feature of HTTP is not enough for your requirements, then you should also evaluate SOAP as you will be building your own cache either way not relying on the protocol.

HTTP Verb Binding -

HTTP verb binding is supposedly a feature worth discussing when comparing REST vs SOAP. Much of public facing API referred to as RESTFul are more REST-like and do not implement all HTTP verb in the manner they are supposed to. For example; when creating new resources, most developers use POST instead of PUT. Even deleting resources are sent through POST request instead of DELETE.

SOAP also defines a binding to the HTTP protocol. When binding to HTTP, all SOAP requests are sent through POST request.

Security -

Security is never mentioned when discussing the benefits of REST over SOAP. Two simples security is provided on the HTTP protocol layer such as basic authentication and communication encryption through TLS. SOAP security is well standardized through WS-SECURITY. HTTP is not secured, as seen in the news all the time, therefore web services relying on the protocol needs to implement their own rigorous security. Security goes beyond simple authentication and confidentiality, and also includes authorization and integrity. When it comes to ease of implementation, I believe that SOAP is that at the forefront.

Why to use Bootstrap and AngularJS?

- REST Easy. RESTful actions are quickly becoming the standard for communicating from the server to the client. In one line of JavaScript, you can quickly talk to the server and get the data you need to interact with your web pages. AngularJS turns this into a simple JavaScript object, as Models, following the MVVM (Model View View-Model) pattern.
- Data Binding and Dependency Injection. Everything in the MVVM pattern is communicated automatically across the UI whenever anything changes. This eliminates the need for wrappers, getters/setters or class declarations. AngularJS handles all of this, so you can express your data as simply as with JavaScript primitives, like arrays, or as complex as you wish, through custom types. Since everything happens automatically,

you can ask for your dependencies as parameters in AngularJS service functions, rather than one giant `main()` call to execute your code.

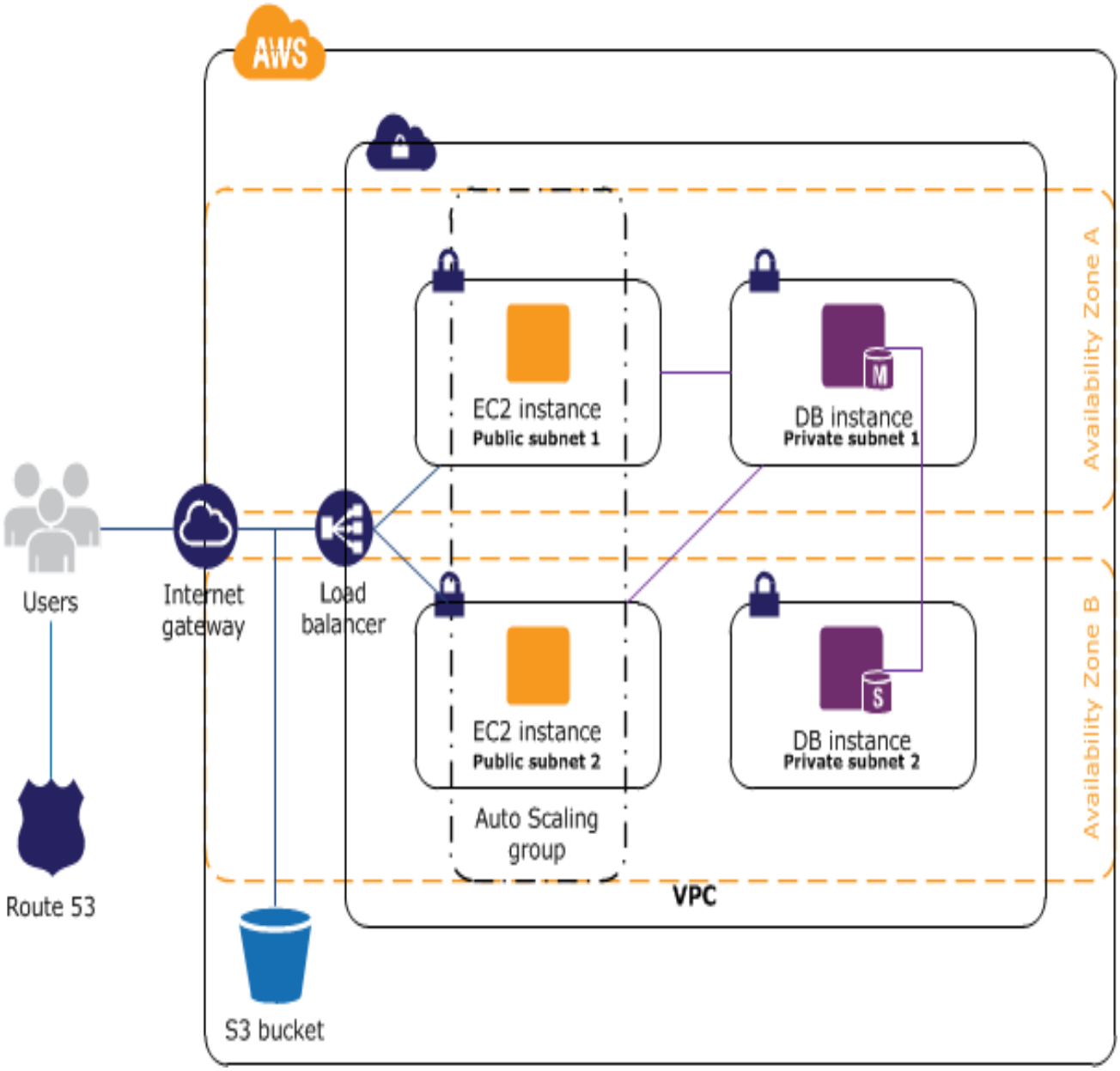
- Extends HTML**.** Most websites built today are a giant series of `<div>` tags with little semantic clarity. You need to create extensive and exhaustive CSS classes to express the intention of each object in the DOM. With Angular, you can operate your HTML like XML, giving you endless possibilities for tags and attributes. Angular accomplishes this, via its HTML compiler and the use ofdirectives to trigger behaviors based on the newly-created syntax you write.
- Makes HTML your Template**.** If you're used to Mustache or Hogan.js, then you can quickly grasp the bracket syntax of Angular's templating engine, because *it's just HTML*. Angular traverses the DOM for these templates, which house the directives mentioned above. The templates are then passed to the AngularJS compiler as DOM elements, which can be extended, executed or reused. This is key, because, now, you have raw DOM components, rather than strings, allowing for direct manipulation and extension of the DOM tree.
- Enterprise-level Testing**.** As stated above, AngularJS requires no additional frameworks or plugins, including testing. If you're familiar with projects, likeQUnit, Mocha or Jasmine, then you'll have no trouble learning Angular's unit-testing API and Scenario Runner, which guides you through executing your tests in as close to the actual state of your production application as possible.

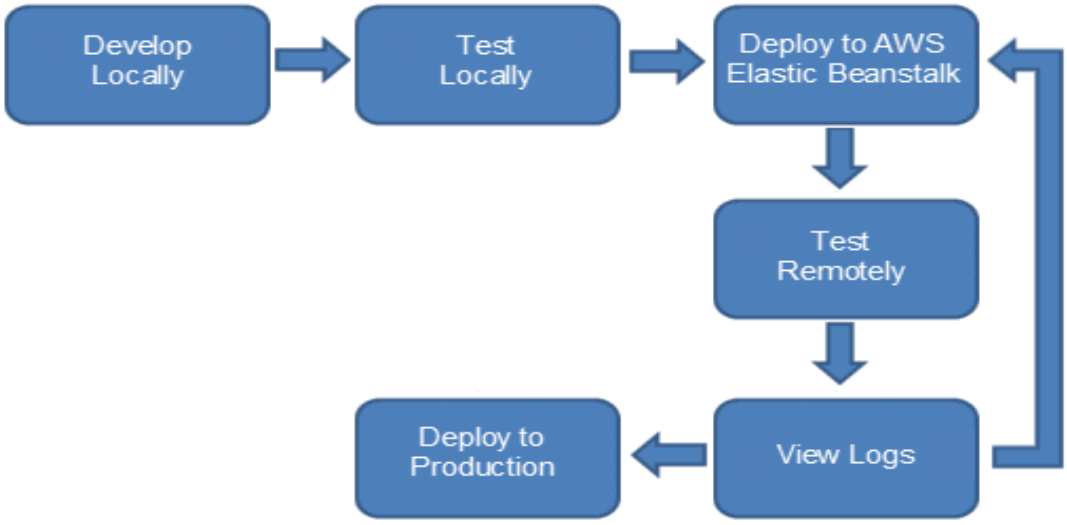
Why web app should be deployed on AWS?

AWS can be used to make it easier to build and manage your websites and applications. The following are some common uses for AWS:

- Store public or private data.
- Host a static website. These websites use client-side technologies (such as HTML, CSS, and JavaScript) to display content that doesn't change frequently. A static website doesn't require server-side technologies (such as PHP and ASP.NET).

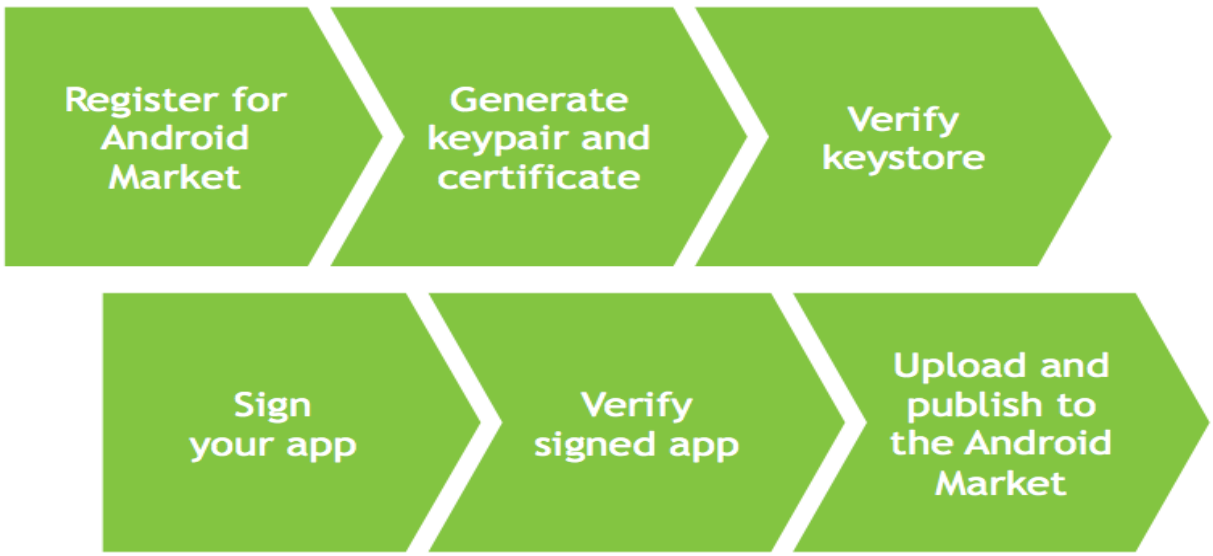
- Host a dynamic website, or web app. These websites include classic three-tier applications, with web, application, and database tiers.
- Support students or online training programs.
- Process business and scientific data.
- Handle peak loads.





Why to deploy Android app on goole play store?

So that it is made available to the users for free. Below are the steps to make it available for the users on google play store –

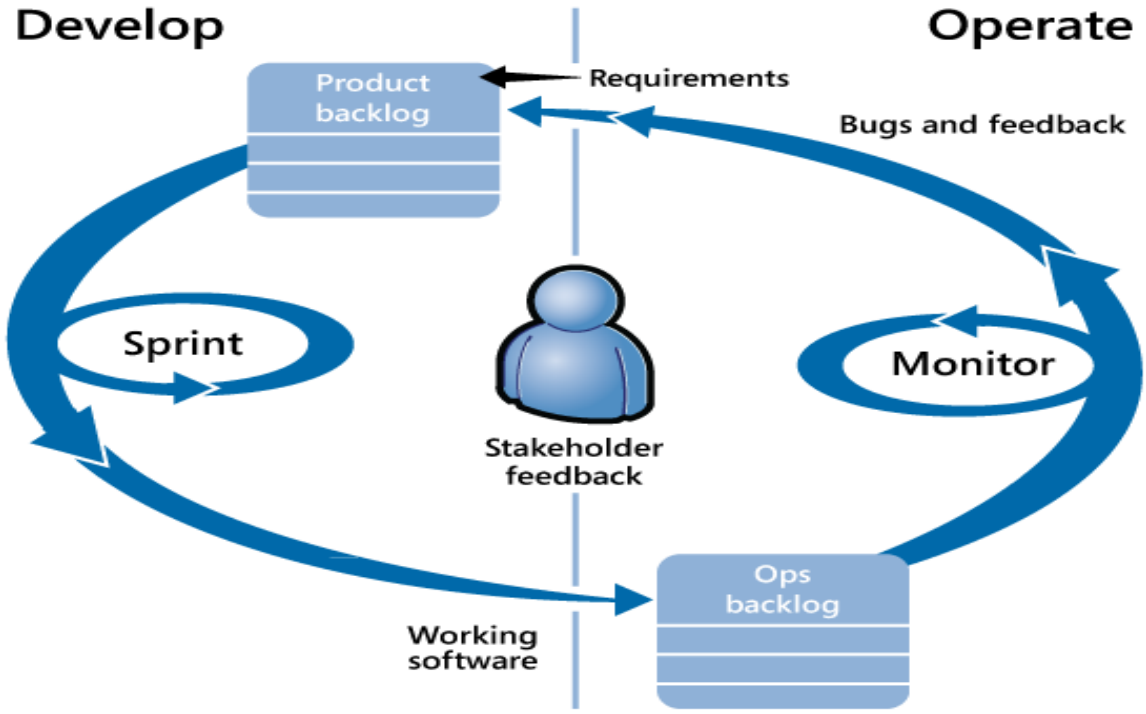


Why is Software testing important?

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.

2. It's essential since it makes sure of the Customer's reliability and their satisfaction in the application.
3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence.
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It's required to stay in the business.



3. System Requirements and Analysis

This project requires the two clusters of 5 systems each, one for stream processing and the other for analytics engine and UI dashboard. Each cluster will communicate with other through high bandwidth network backbone to support uninterrupted massive message transfer. We are planning to create all the required virtual machines on 2 high configuration machines that would form the clusters. As part of the software requirements we would use the open source libraries of apache Kafka and apache Storm, esper and all the available e-commerce merchant api.

3.1 Domain and business requirements

Business Modeling

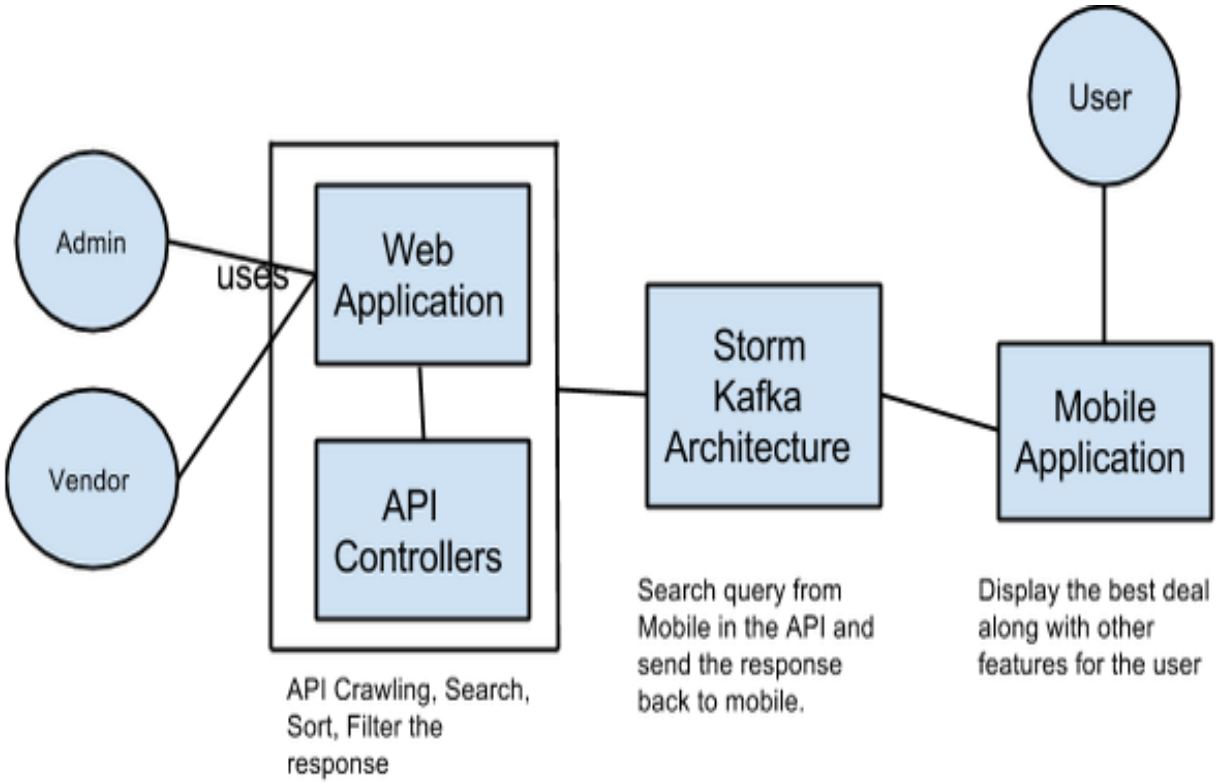


Figure 3.- PriceKing Business Model

Business requirement of our system is that it should be able to provision the businesses to post their products to user's cell phone and it should also keep track of the use's searches and analyze user's likes, preferences and search patterns.

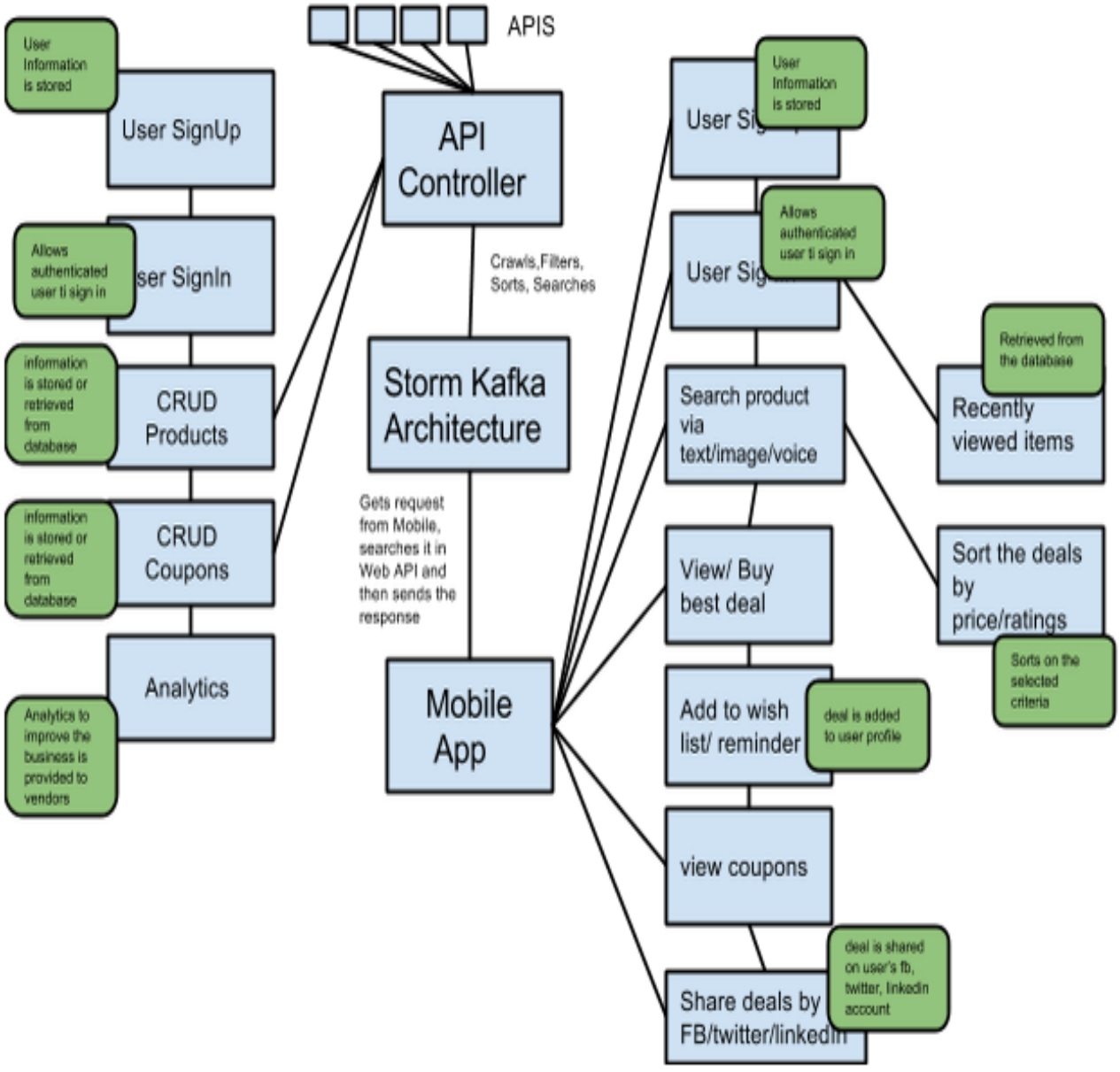


Figure 3.- PriceKing business model

Businesses should be able to provide custom promotions for the prospective buyers and the system should be able to provide the reports of the business done in weekly, monthly and quarterly fashion.

Class Diagram:

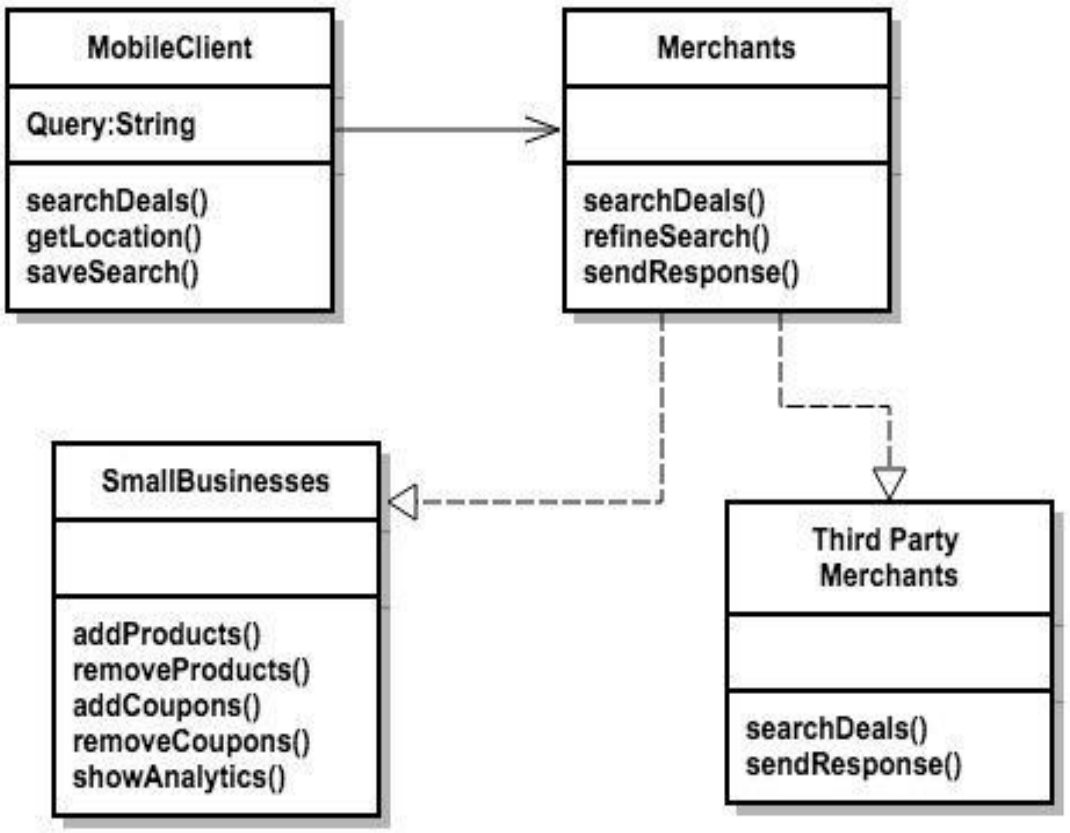


Figure 3.1: Class Diagram

Activity Diagram:

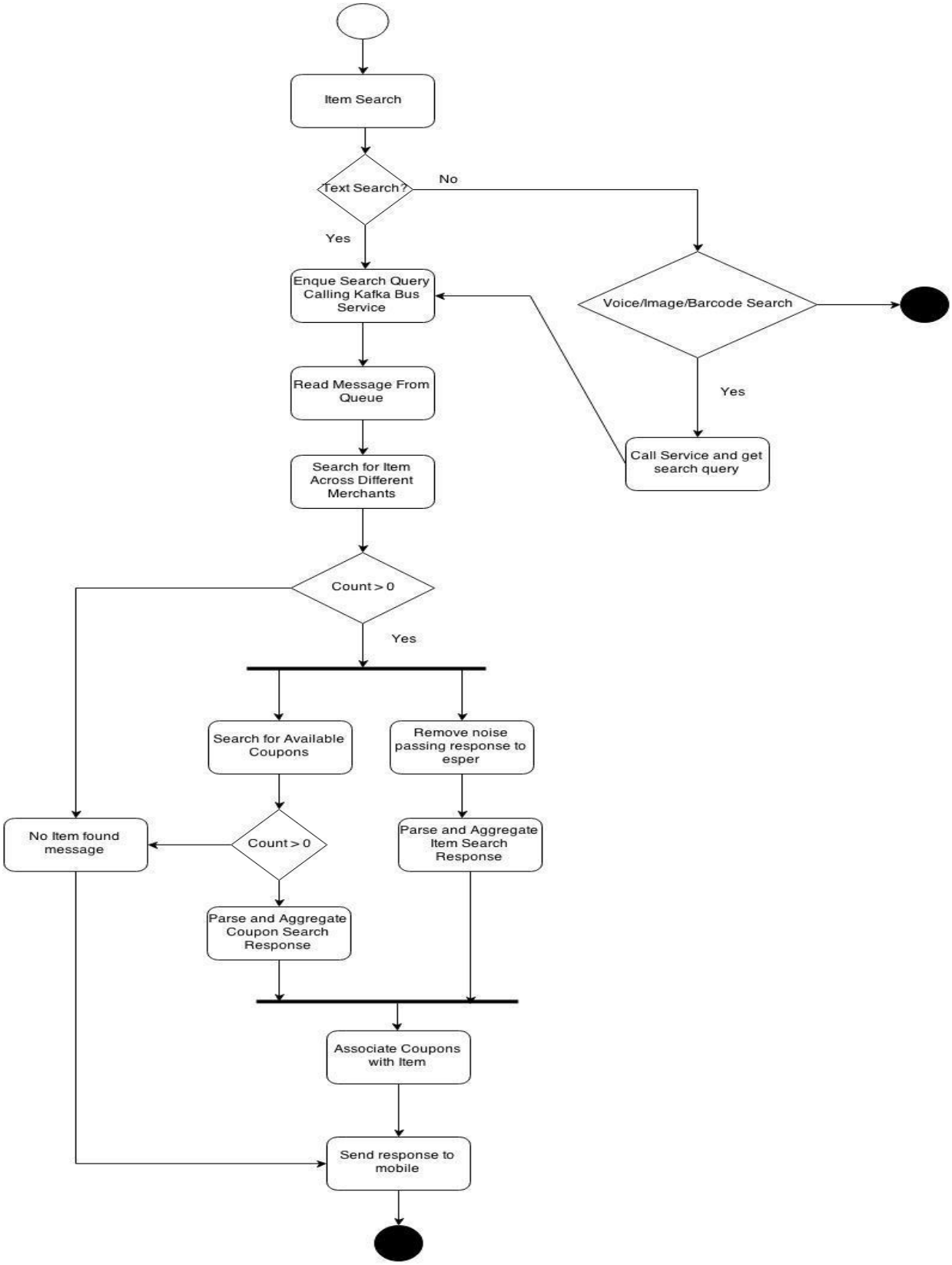


Figure 3.2: Activity Diagram Of The System

3.1.1 Process Decomposition Diagram

3.1.1.1 Search User Query

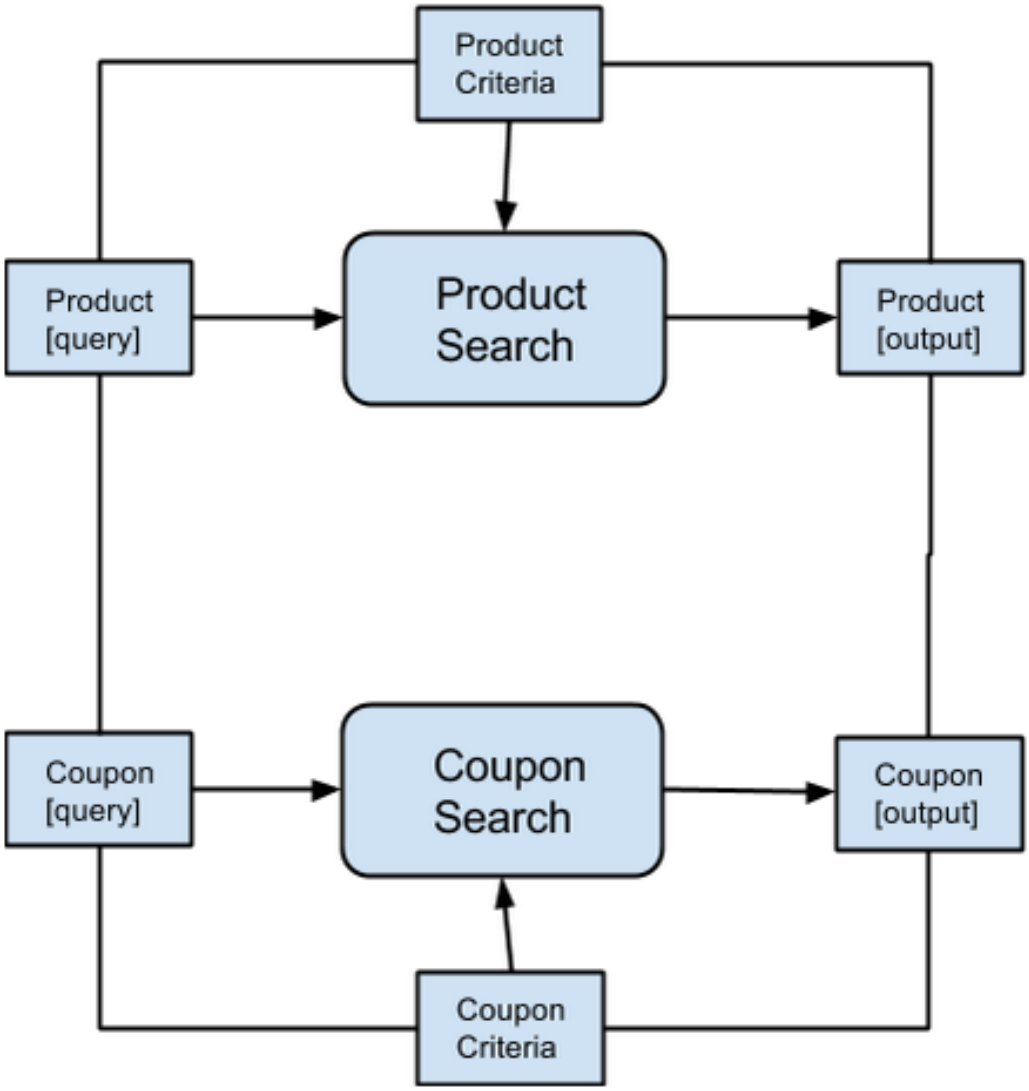


Figure 3.3 Search User Query

Search User Query process will have two sub-processes Product Search and Coupon Search, Product Search sub-process searches products on BestBuy, eBay and Walmart as well as registered small businesses . Coupon Search sub-process searches coupons and daily deals on GroupOn and registered small businesses.

3.1.1.2 User Preference

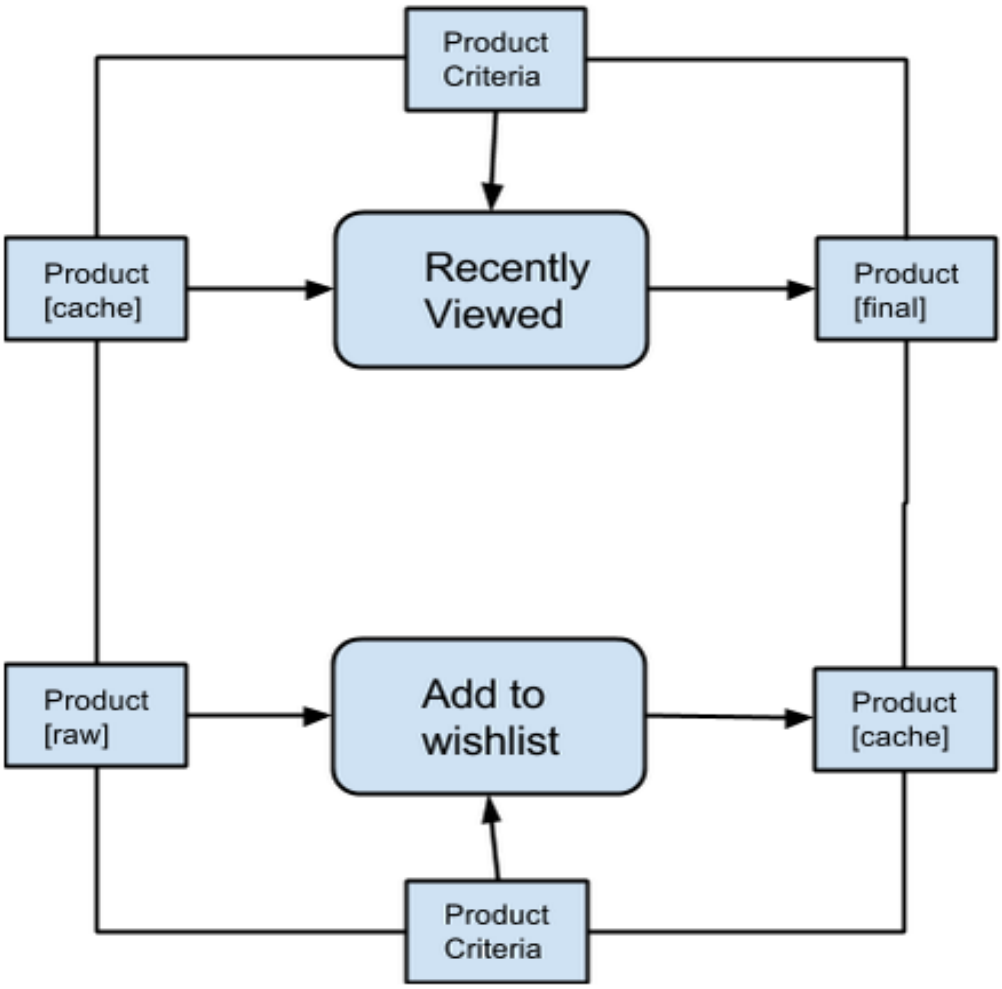


Figure 3.4:User Preferences

User preferences process will have two sub-processes Recently Viewed and Add to Wish list. Recently Viewed sub-process will keep the track of recently viewed items by user. Add to wish list will save product/coupon as a wish list for future purchase.

3.1.1.3 Social Media Activity:

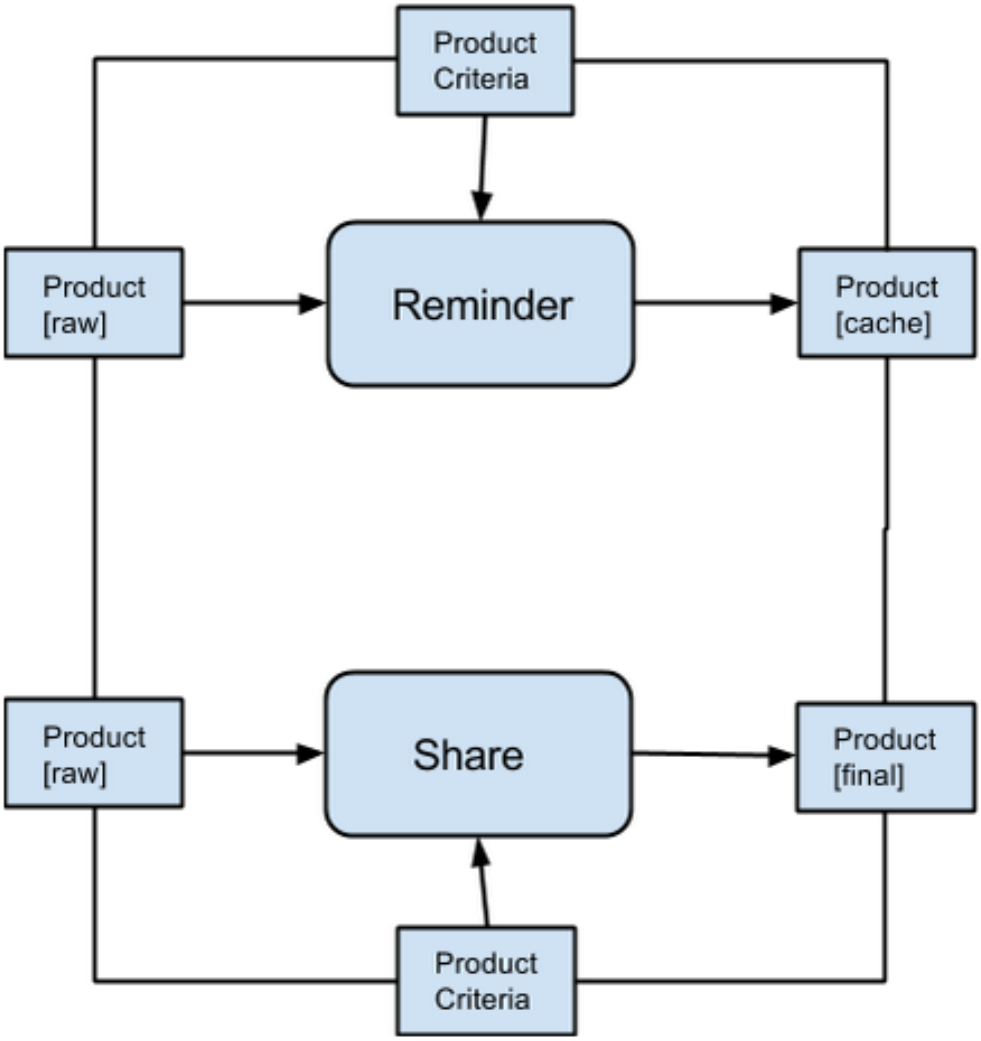


Figure 3.5:Social Media Activity

Social media activity will have two sub-processes, Reminder and Share. Reminder will set a reminder to purchase a product on a preferred date and time. It will notify user using push notification service. Share sub-process will allow users to share product/coupon on a social media websites like Facebook, twitter and LinkedIn.

3.1.2Package Diagram

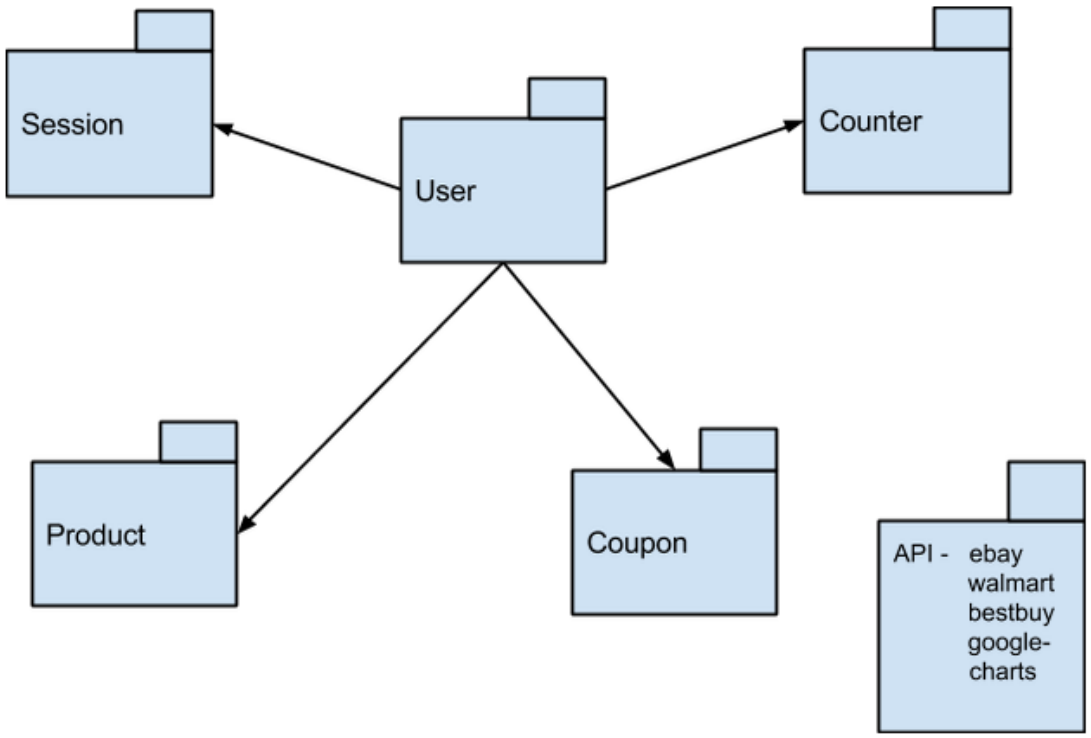


Figure 3.6 Package Diagram

3.2 Customer-oriented requirements

The customer of this application is a common person who wants to get the best deals on his purchases. The user should be able to see the deals on his set preferences. The customer of this application should be able to save his searched deals and should be able to share his searches on social media.

The application should serve Android based mobiles and tablets as client. It means the application should support all resolutions of Android devices such as LDPI, MDPI, HDPI, XHDPI, XXHDPI and XXXDPI. The interactive User Interface should make the app look beautiful and keep the user engaged. The application should enable user to share the cheapest deal found on

various social networking platforms such as Facebook, Twitter, etc. Moreover the app should display advertisements of the products based on users interests and searching patterns either through push notifications or directly from the app using Google Mobile Adds. The app should make use of SQLite database, Shared Preferences for client side storage. The app should make use of Google Maps V3 for the geolocation analysis. Implementation of Google analytics V4 would help us target our application visitors and measure the results of our advertising/marketing campaign. The app should provide multiple ways of user inputs (text, voice, image) based on user interest. The app should allow user to add any specific deal to a calendar as reminder.

There are three types for users for both the web and android application- admin, end users and small business vendors.

- End users can search for a particular product via text/image/voice and will get the best deal. Users can sign in, signup, share the deal on social media like Facebook, twitter or LinkedIn and add reminder on the calendar.
- Small business vendors can sign up, sign in, add/delete/update products/coupons, get analytics for improving their business.
- Admin can have insight of everything.

System level use case diagram

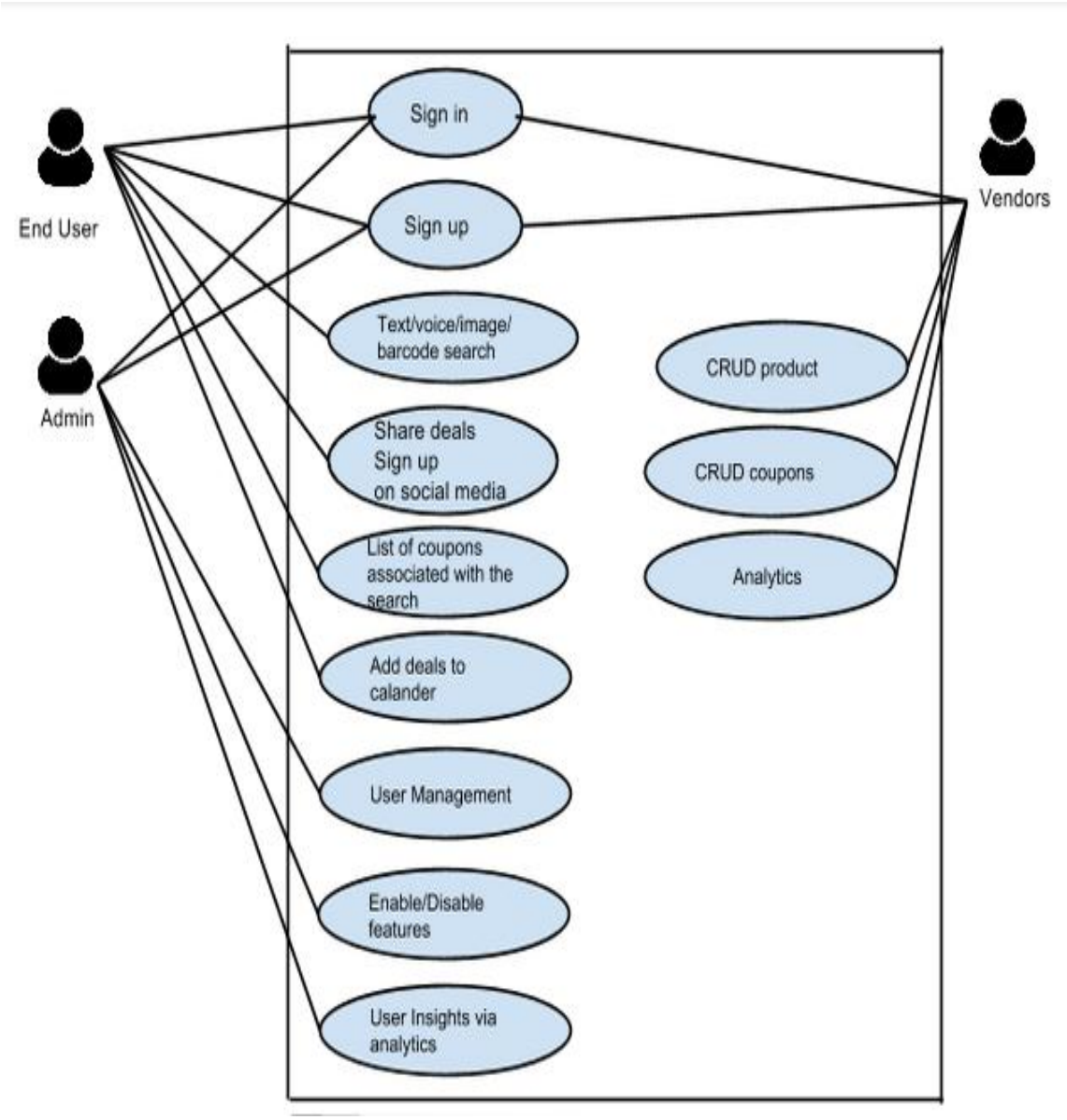


Figure 3.7: System level use case

<table>
<tr><td>Use Case Id</td><td>PriceKingUseCase-01</td></tr>
<tr><td>Name</td><td>PRICEKING-UC-EndUser</td></tr>
<tr><td>Actors</td><td>End User</td></tr>
<tr><td>Description</td><td>Can search for a particular product via text/image/voice and will get the best deal. Users can signin, signup, share the deal on social media like facebook, twitter or linkedin and add reminder on the calendar.</td></tr>
<tr><td>Pre-condition</td><td>Mobile UI, Data from APIs in common format required</td></tr>
<tr><td>Post-condition</td><td>None</td></tr>
<tr><td>Main Flow</td><td>
<table>
<tr><td>Mobile App</td><td>System</td></tr>
<tr><td>1)LogIn and Search for a product via text/image/voice</td><td></td></tr>
<tr><td></td><td>2) Sends request to the App controller
3) Controller searches for a particular product by crawling APIs
4) Gets the result, sorts it and sends it to mobile app</td></tr>
<tr><td>5) Gets the result
6) Buy product/add it to wish list, set reminder, share it on facebook/linkedin/twitter.</td><td></td></tr>
<tr><td>7) Visualize analytics</td><td></td></tr>
</table>
</td></tr>
<tr><td>Special Requirements</td><td>None</td></tr>
</table>

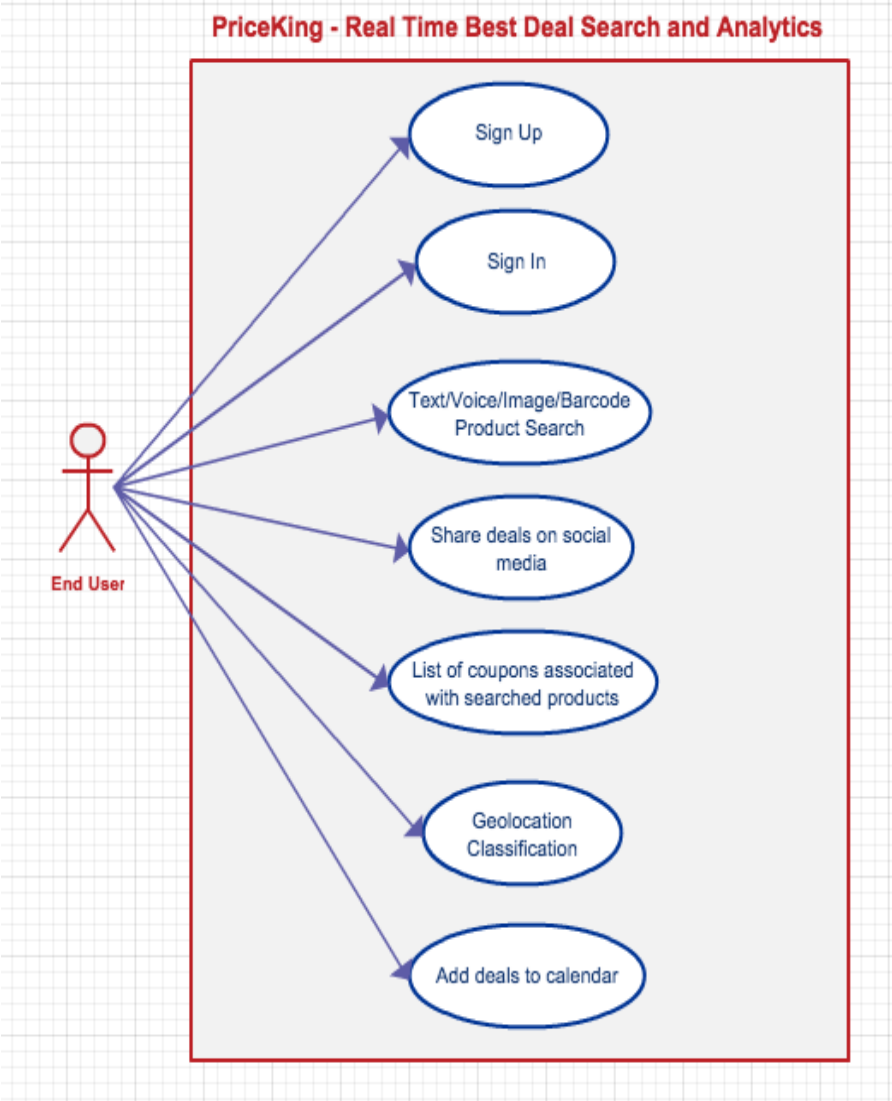


Figure 3.8: Use case Diagram for End User

Table 3.- Small business use case table

Use Case Id	**PriceKingUseCase-02**
Name	**PRICEKING-UC-Vendors**
Actors	**Small Business User**
Description	Can sign up, sign in, add/delete/update products/coupons, get analytics for improving their business.
Pre-condition	Web UI, Database dore storage and retrieval
Post-condition	None

<table>
<tr><td>Main Flow</td><td>
<table>
<tr><td>Web App</td><td>System</td></tr>
<tr><td>1)LogIn and add/edit/delete products/coupons</td><td></td></tr>
<tr><td></td><td>2) Performs add/edit/delete and updates database</td></tr>
<tr><td></td><td>3) Plots analytics graphs</td></tr>
<tr><td>7) Visualize analytics and market product/coupon</td><td></td></tr>
</table>
</td></tr>
<tr><td>Special Requirements</td><td>None</td></tr>
</table>

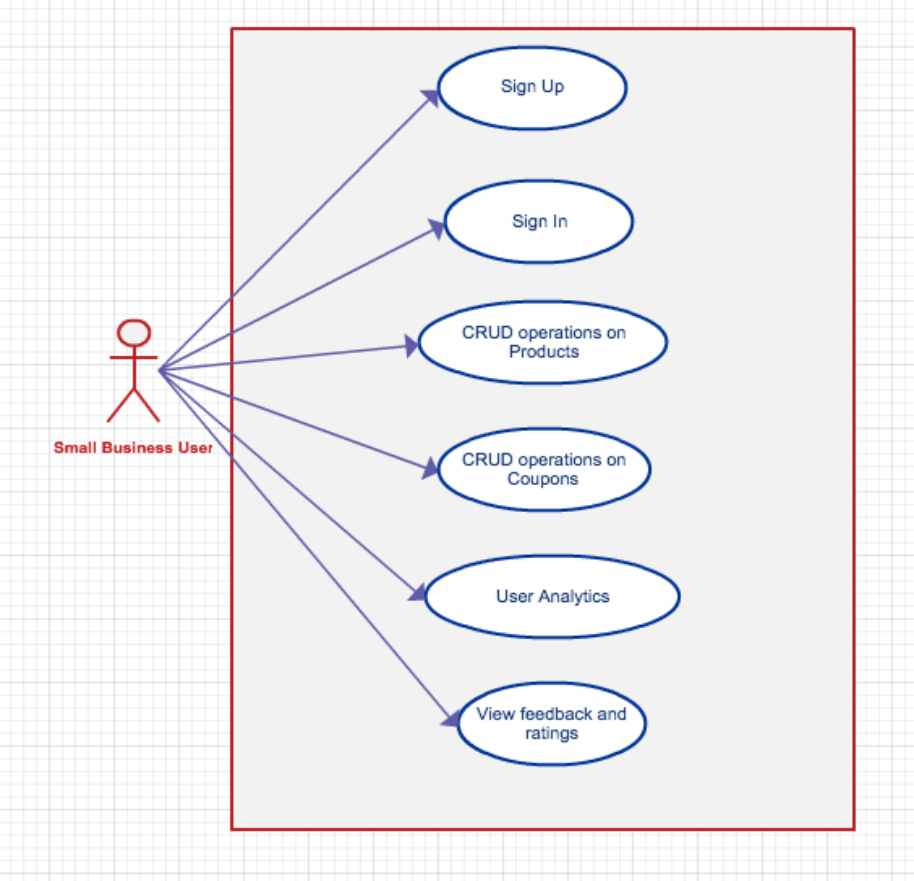


Figure 3.9: Use case Diagram for Small Business User

Table:3.- Admin Use case

<table>
<tr><td>Use Case Id</td><td>PriceKingUseCase-03</td></tr>
<tr><td>Name</td><td>PRICEKING-UC-Admins</td></tr>
<tr><td>Actors</td><td>Admin</td></tr>
<tr><td>Description</td><td>Can get insight for vendor’s activities.</td></tr>
<tr><td>Pre-condition</td><td>Web UI, admin credentials</td></tr>
<tr><td>Post-condition</td><td>None</td></tr>
<tr><td>Main Flow</td><td>
<table>
<tr><td>Web App</td><td>System</td></tr>
<tr><td>1)LogIn and view all the users
2)look for products/coupons</td><td></td></tr>
<tr><td></td><td>2) Fetch it from database and send response.</td></tr>
<tr><td>3) Get the details and analytics</td><td></td></tr>
</table>
</td></tr>
<tr><td>Special Requirements</td><td>None</td></tr>
</table>

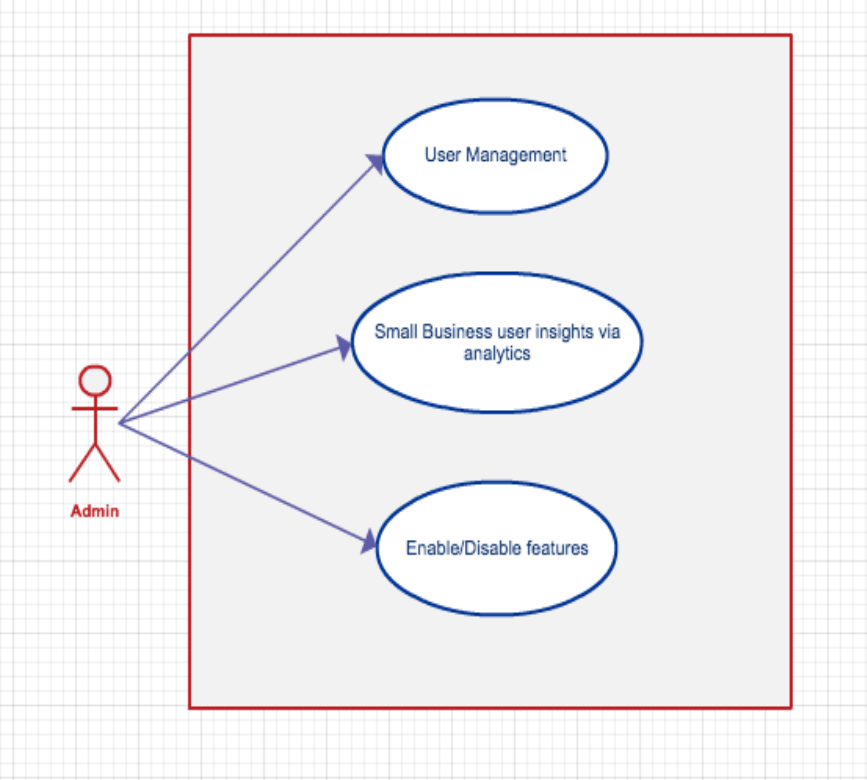


Figure 3.10 Use case for Web UI and Dashboard Admin

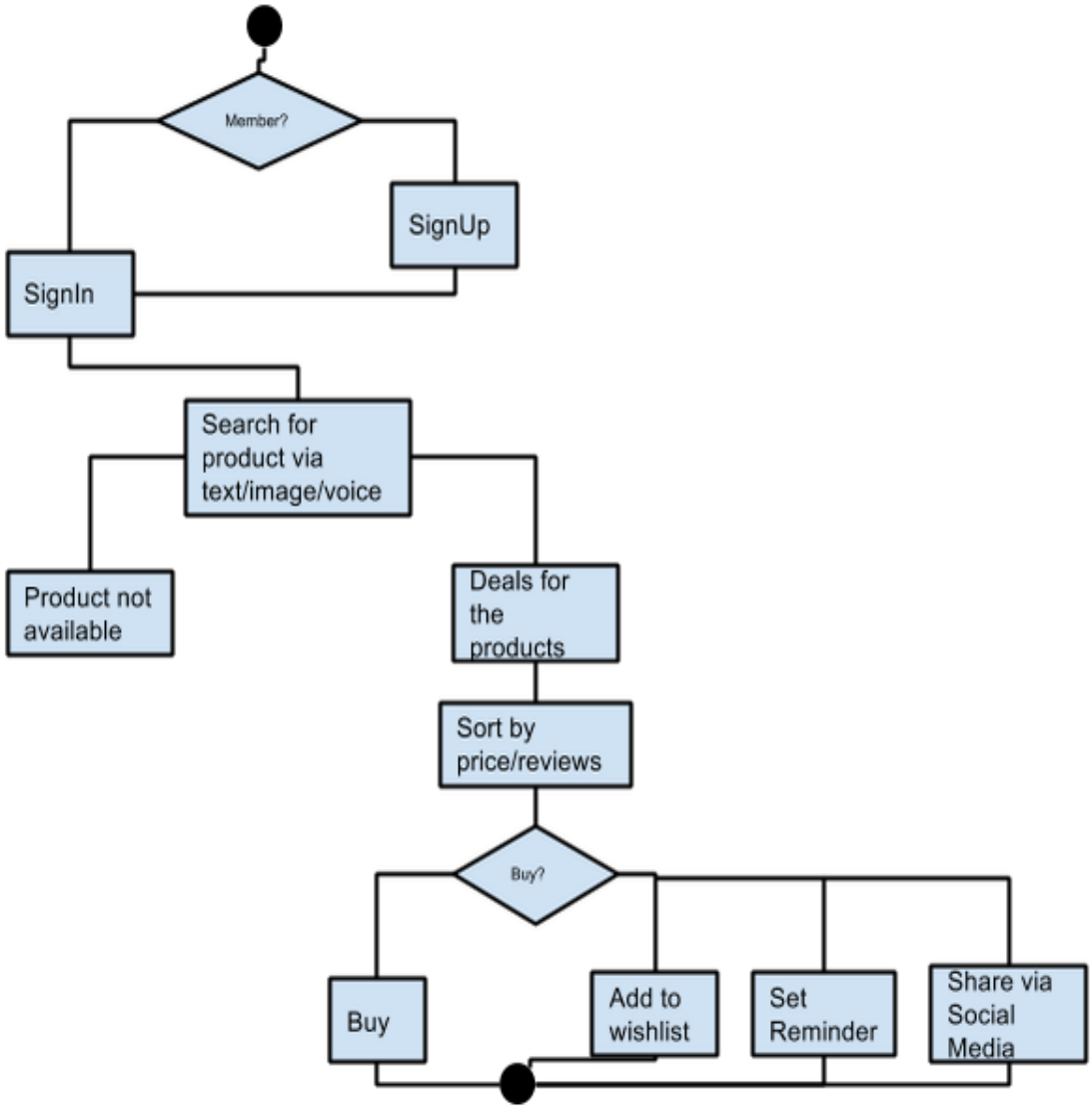


Figure 3.11:User state machine diagram

3.3 System functional requirements

Search query string from mobile type ahead API should be sent to the middleware i.e. Kafka Storm topology. The item in request should be searched on the entire available vendor API and the results should be sent to aggregator. The aggregator should collect results from different API's and mold them into common message format. The aggregated results should be passed to esper engine which should remove unwanted results based on user preferences and should send the filtered results back to the user.

3.4 System behavior requirements

Sequence Diagram: Mobile User

Below is the sequence diagram in which whole mobile app is well captured. When a user searches for a particular product, search query is sent to the server and it is sent to the third party merchants i.e. through crawling their APIs. Then search response is sent to the server. The server filters the response, sorts it and sends it to the mobile UI.

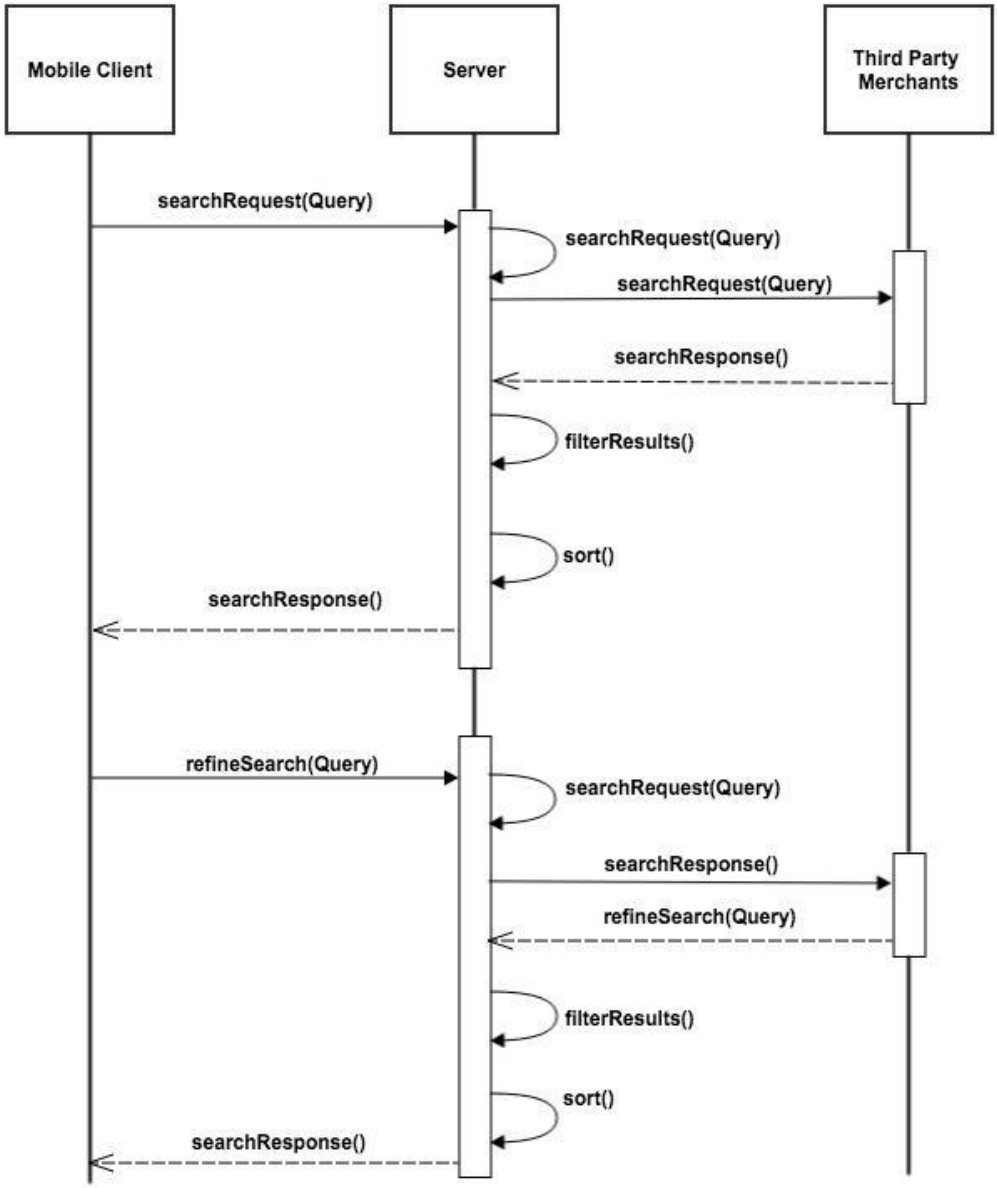


Figure 3.12: Sequence Diagram Mobile User

Sequence Diagram: Small Business

The sequence diagram below exhibits the flow of Web application. When a vendors/small business user adds a product/coupon, the server processes the request and sends the failure/success message in response. Server processes all the data and shows the big picture to the users in form of analytical charts. Admit can have insight to all the activities.

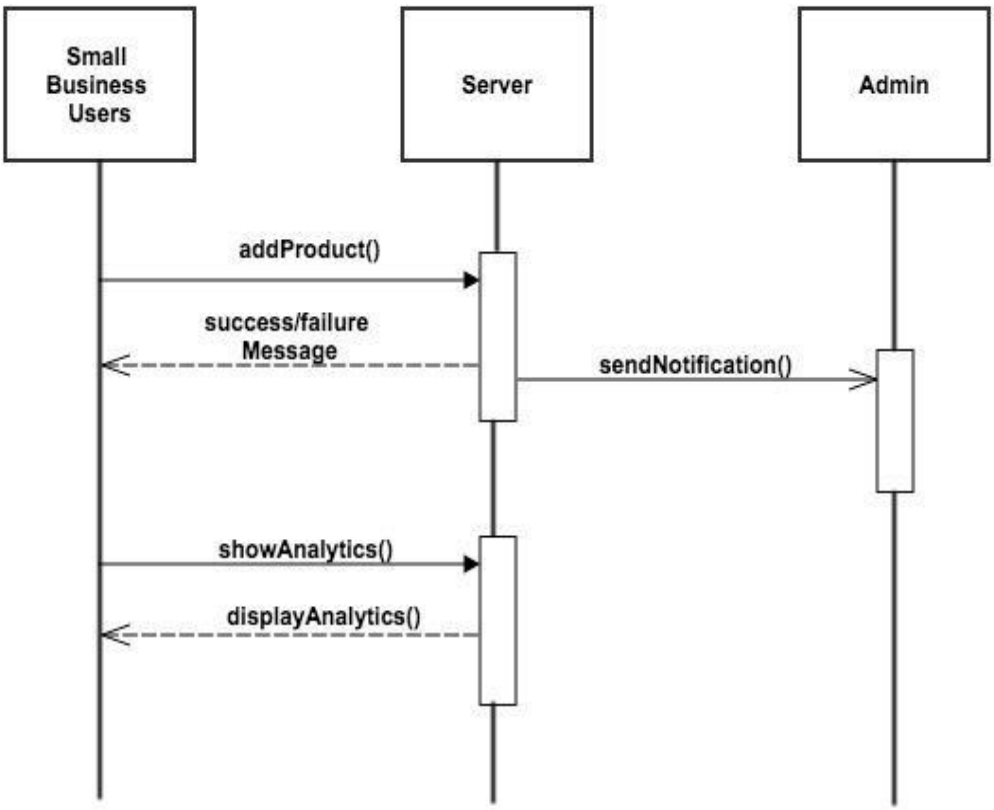


Figure 3.13: Sequence Diagram Small Business

1. Initially the user hits specific UI elements that trigger the web service, which runs on a different thread in the background. It's a synchronous call where the user interface displays a progress dialog until the callback is received.

2. This service then connects to our backend server through HTTP request to get the list of products based on user search and returns with the JSON response and a status code.

3. If the status code is 200 (SUCCESS) then the JSON parsing is executed on the received response and the data is inserted in the storage, which is SQLite database in this application.

4. Once the data is stored in DB, the control goes to service, which returns the control to the user interface through a handler. The control then goes to the listener success method on the user interface where the progress dialog was shown in step 1. This progress dialog is then dismissed and the user interface is filled with the data retrieved from the SQLite database.

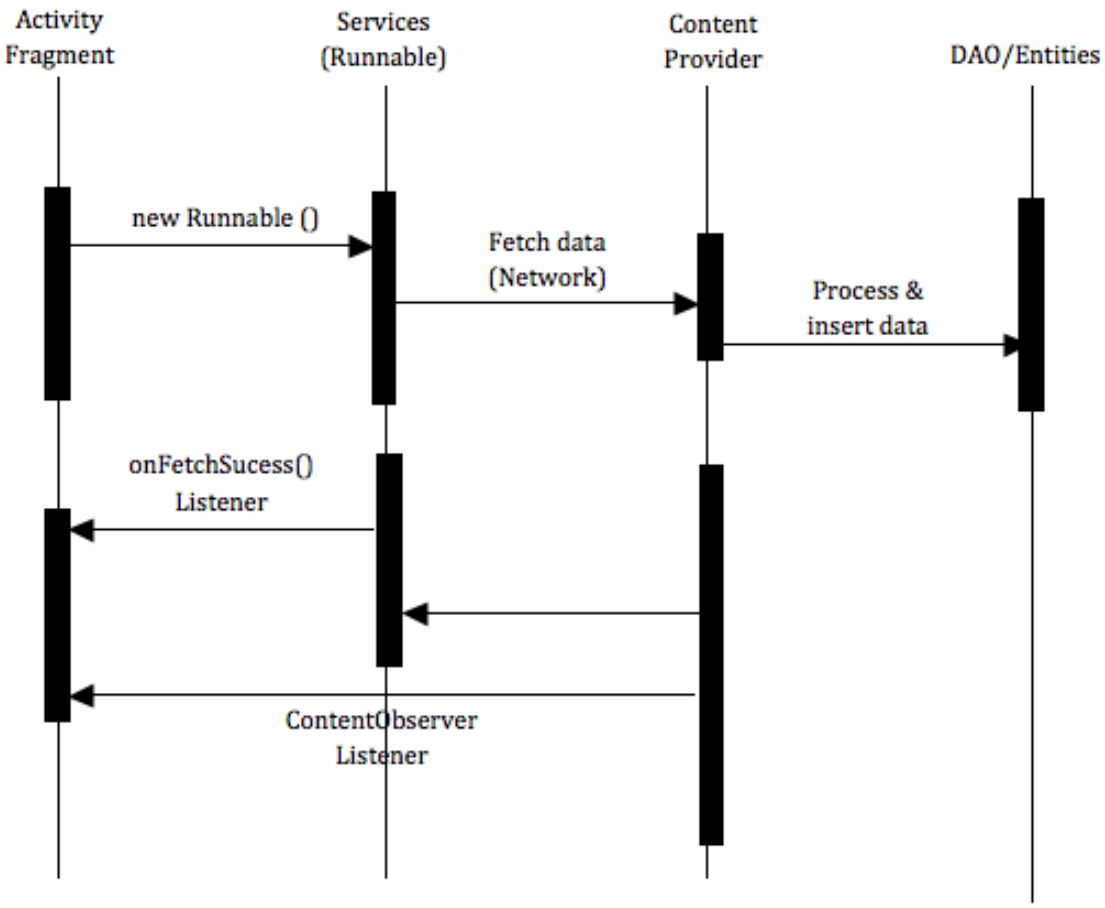


Figure 3.14: Sequence diagram for PriceKing Android App

3.5 System performance and non function requirements

Requirements/ Criteria	Description
Performance	The performance of the system should be high and it should be able to serve 100 concurrent users without any delay in service based on our initial infrastructure.
Capacity	The mobile application should be able to display 8-10 deal searches in one screen depending on the resolution of the phone. The backend should be able to crunch 1000 requests concurrently.
Availability	The services should be available with average downtime of 100 hours per year depending on the hardware quality and resource availability.
Security	The mobile application should be secured and adhere to android security standards and Web and UI dashboard should be secured from unauthorized access and cross side scripting and all other malicious attacks and vulnerabilities.
Compliance to Standards	The mobile application as well as Web application should be compliant to the CMMI-DEV-v1.2, ISO/IEC and IEEE/EIA Standards 12207, IEEE/EIA Standard 1058, IEEE 830 standards.

Table 3.1 Performance and Non-Function Requirements

3.6 System context and interface requirements

The System would be built in different environments like, the android app will be built in ADT, the middleware services would be written in java and UI dashboard would be written in HTML5 Angular.Js and jsp. All the application services would be deployed on cloud and android app would be available on Google play.

Mobile App	Deployed on Google play, built in Android Development Toolkit
Middleware	Cloud deployed services, built in java
Web application	Cloud deployed application built in Angular.js and Jsp

Table 3.2 Context and Interface Requirements

3.7 Technology and resource requirements

Hardware Requirements: 10 medium sized virtual machines, 1 network switch. Software Requirements: Binaries for Kafka Web Console, Kafka, Storm, esper, Mongodb, Storm UI, spring, ADT, Angular.js.

Your Group #:					
Your Group Name:	Maverics				
Project Factors			**Organization Factors**		
Product Size	202	ESUs	**Development Effort Percentages**		
Schedule Constraint	25.25	Weeks	Analysis & Design Effort	25	%
Loaded Salaries	$1,000	$ per SW	Software Construction Effort	60	%
			Integration and Test Effort	15	%
Estimation Adjustment Factors			**Development Schedule Percentages**		
CPLX (Product Complexity	1.2		Analysis & Design Schedule	30	%
AXEP (Application Experie	1.1		Software Construction Schedule	50	%
PCAP (Programmer Capat	1.2		Integration and Test Schedule	20	%
TOOL (Use of Software To	0.9				
MODP (Use of Modern Pro	0.8	EAF =	1.1		
			EAF should be in the range of 0.8 to 1.2;		
Development Estimates					
Development Effort	65	SW	60% of total effort		
Development Schedule	19	Weeks	75% of constrained schedule		
Development Cost	$64,799	Dollars			
Development Effort Breakdown			**Development Schedule Breakdown**		
Analysis & Design	16	SW	Analysis & Design	5.6813	Weeks
Software Construction	39	SW	Software Construction	9.4688	Weeks
Integration and Test	10	SW	Integration and Test	3.7875	Weeks
			Total Development Schedule	18.938	Weeks
Development Staffing Level					
Analysis & Design	3	People			
Software Construction	4	People	**Additional Schedule Breakdown**		
Integration and Test	3	People	System V&V	4.2168	Weeks
			Installation	2.1084	Weeks
			Project Mgmt, CM, and Tech. Pubs.	25.25	Weeks
TOTAL PROJECT EFFORT					
Development Effort	65	SW			
Additional Effort	43	SW	**Additional Effort Breakdown***		
Total Effort	108	SW	Do Project Management	11	SW
			Verify and Validate System	9	SW
TOTAL PROJECT SCHEDULE			Perform CM	5	SW
	25	Weeks	Prepare Tech. Pubs.	5	SW
TOTAL PROJECT COST			Install System	2	SW
Development Cost	$64,799				
Additional Cost	$43,415		**Additional Staffing Breakdown**		
Total Cost	$108,214		Do Project Management	0.4	People
			Prepare Tech. Pubs.	0.2	People
			Perform CM	0.2	People
			Verify and Validate System	2.1	People
			Install System	1.0	People

Table 3.3: Resources Estimate

4. System Design

4.1 System Architecture Design

The architecture of the system can be shown below. The high level architecture of our system consists of mobile as a client Kafka-Storm topology as a middleware and Web and analytics engine as a UI for small businesses and analytics dashboard for business users.

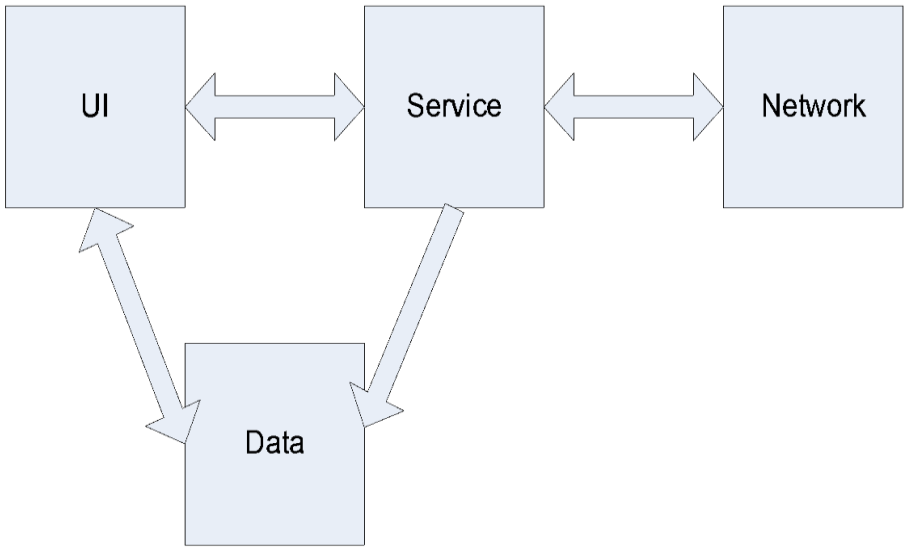


Figure 4.1: Mobile and Tablet Application

High-level architecture for this mobile Android app is based on MVC (Model View Controller) design pattern. In this architecture the user requests from view and a service is generated that is managed by controller and eventually returns the result list of Products that is stored by model in the SQLite database. The controller runs each service on a different thread that runs in the background. The user is shown a progress dialog till the response is returned to the different parts of UI as shown in the high-level architecture.

Below is the big picture of middleware architecture Kafka maintains a partitioned log for each topic which is sequential immutable collection of messages where every new message is

appended to the end of it and each message in the partition is uniquely identified by an offset. Storm runs the topologies and in esper bolts we have queries stored and the data is flowing.

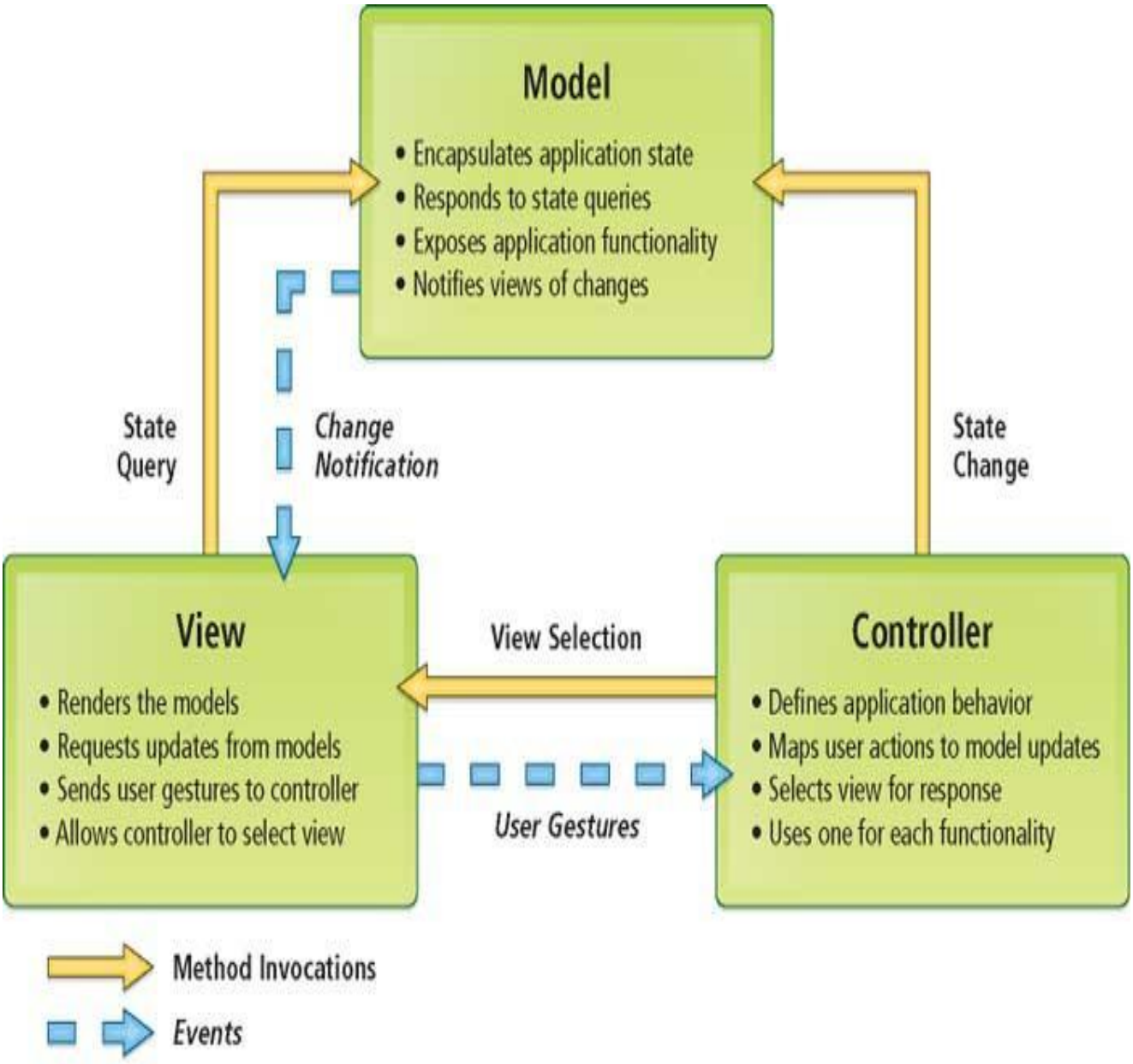


Figure 4.2:MVC Architecture

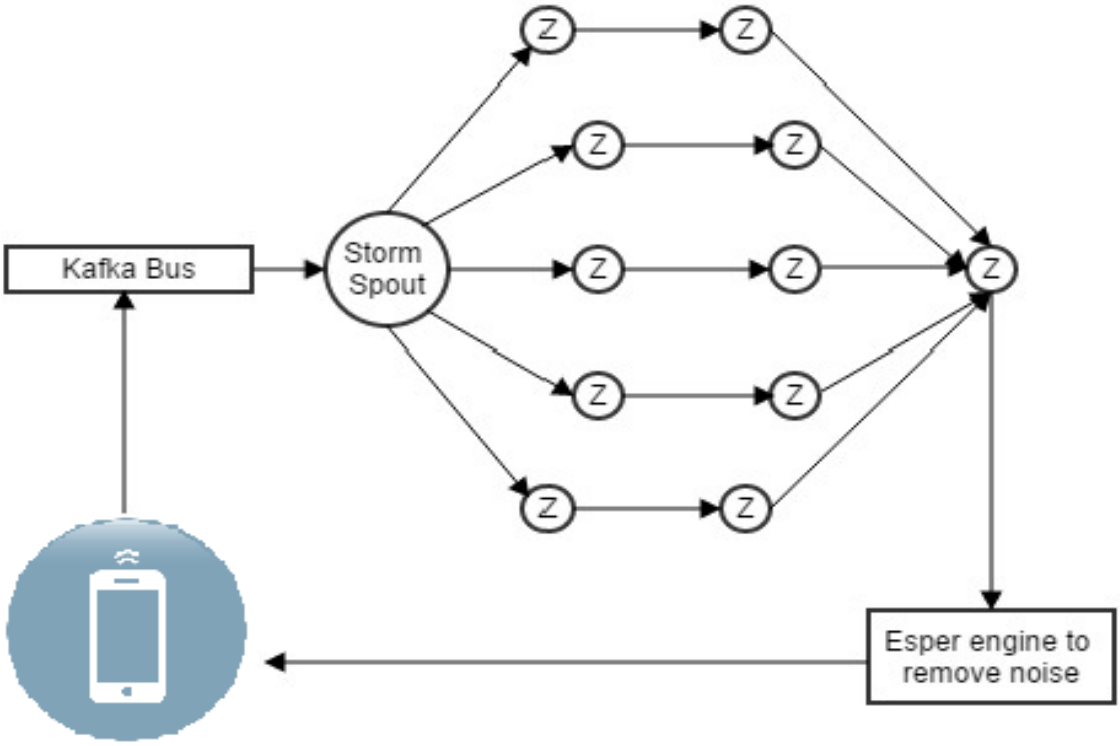


Figure 4.3: Middleware Services

4.2System Data and Database Design

The database schema is still under discussion and collection structures is yet to be finalized however, system data will be in the form of JSON and the database would be Mongodb.

Collections

```
> show collections
counters
coupons
products
profiles
session
system.indexes
```

Session _Id Status	**Counters** _Id SequenceValue
Profiles _id firstname lastname email phone password	**Products** _id productName productDescription productCategory thumbnailImage productUrl price uploadedBy
Coupons _id name description originalPrice discountedPrice couponCode expiryDate	

Table 4.1: Mongo Collections

The counter collection keeps the track of number of vendors, products and coupons through their respective ids

Counters

```
> db.counters.find().pretty()
{ "_id" : "couponId", "sequenceValue" : 13 }
{ "_id" : "productId", "sequenceValue" : 24 }
{ "_id" : "vendorId", "sequenceValue" : 1 }
```

Product collection

The product collection stores the details of the products like name, description, category, etc. Id is generated by auto-increment and the count is stored in counters collection.

```
> db.products.find().pretty()
{
        "_id" : 23,
        "productName" : "iPhone6",
        "productDescription" : "N/A",
        "productCategory" : "Electronics",
        "thumbnailImage" : "",
        "productUrl" : "N/A",
        "price" : "634",
        "uploadedBy" : "hardik",
        "isDeleted" : false,
        "productDesctiption" : "N/A"
}
```

Coupon collection

The coupon collection stores the details of the coupons like name, description, expiry date, etc. Id is generated by auto-increment and the count is stored in counters collection.

```
> db.coupons.find().pretty()
{
       "_id" : 13,
       "name" : "iPhone6 Case",
       "description" : "Compatible with Apple iPhone 6; TPU material; clear design; fit construction",
       "originalPrice" : 27,
       "discountedPrice" : 13,
       "couponCode" : "GHGJ890H",
       "category" : "Electronics",
       "expiryDate" : ISODate("2015-05-09T07:00:00Z"),
       "uploadedBy" : "hardik",
       "isDeleted" : false
}
```

Users(profiles) collection

The profiles collection stores the information about users like name, email id, password, etc. Id is generated by auto-increment and the count is stored in counters collection.

```
> db.profiles.find().pretty()
{
        "_id" : "hardikjoshi",
        "password" : "ZwppQxfvv73vv71gXe+/ve+/vR9bWWbvv70=,-1659087522",
        "firstname" : "hardik",
        "lastname" : "joshi",
        "email" : "hardikjoshi.se@gmail.com",
        "phone" : "000-000-0000"
}
```

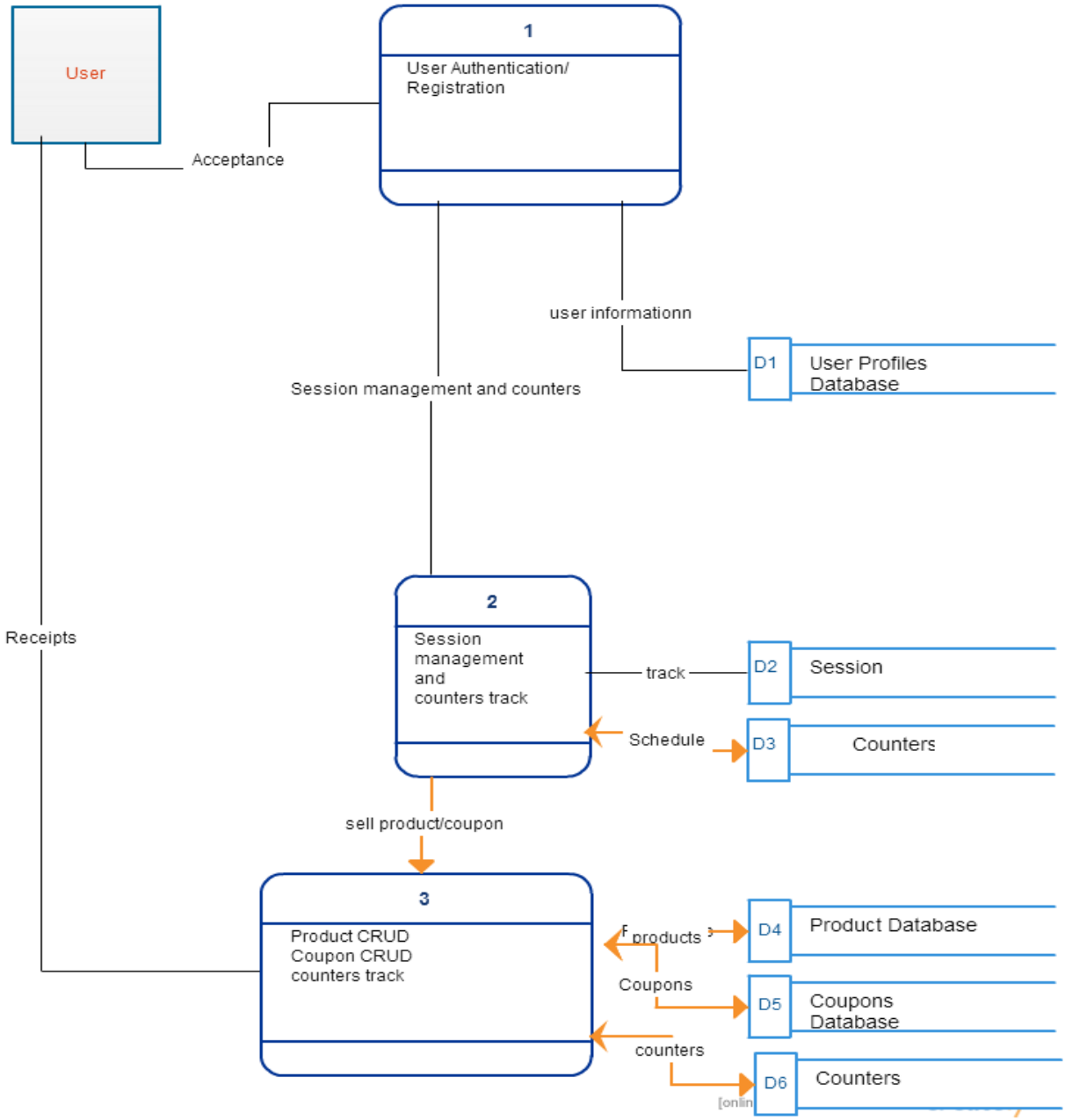


Figure 4.4: Data flow diagram

The above diagram shows the data flow in PriceKing Dashboard application. In first step user is required to follow the authentication process, once authenticated, user establishes a session and can perform CRUD operation on products and coupons. The added coupons and products are stored in their respective collections and the proper count is maintained for the added products and coupons.

Session collection

The session collection is used for maintaining the active/inactive sessions.

```
> db.session.find().pretty()
{
        "_id" : "hnUhC1S0I2DKqrmRjXadK0WZTMsqdrSxTmLxZcGnfKE=",
        "username" : "devenpawar18"
}
{
        "_id" : "p+S6N0keIRc1dfm1N20KpGp10N4l1wVnZ66aoq16jwU=",
        "username" : "hardik"
}
```

Below is the mongodb config file. A common file that contains the mongo configurations so that while performing database operations one just needs to call this configuration file.

```java
package Config;

import java.net.UnknownHostException;
import com.mongodb.MongoClient;

public class MongoConnection {
    public static MongoClient getNewMongoClient()
    {
    MongoClient mongoClient  = null;
    try {
    mongoClient = new MongoClient(PricekingConfig.SERVER, PricekingConfig.PORT);
    } catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
    }
    return mongoClient;
    }
    public static void closeMongoClient(MongoClient mongoClient)
    {
    mongoClient.close();
    }
}
```

```java
package Config;

public class PricekingConfig {
    public static final String SERVER = "localhost";

    public static final int PORT = 27017;

    public static final String DBNAME = "priceking";
    public static final String PROFILECOLLECTION = "profiles";

}
```

4.-MongoDB Config Code Snippet

Schema for Android App:

There are 4 tables in the Android application schema.

- Product - This table consists of user searched results of any product.
- Recently Viewed - This table consists of products that were recently searched by the user that tracks user history.
- Wish List - The app allows user to add any product to the user's wish list that the user might want to buy later. In order to add a product to wish list, a user must be signed in the app.
- Advertisement - This table consists of advertisements of different product of user's interest on the home screen of the app.

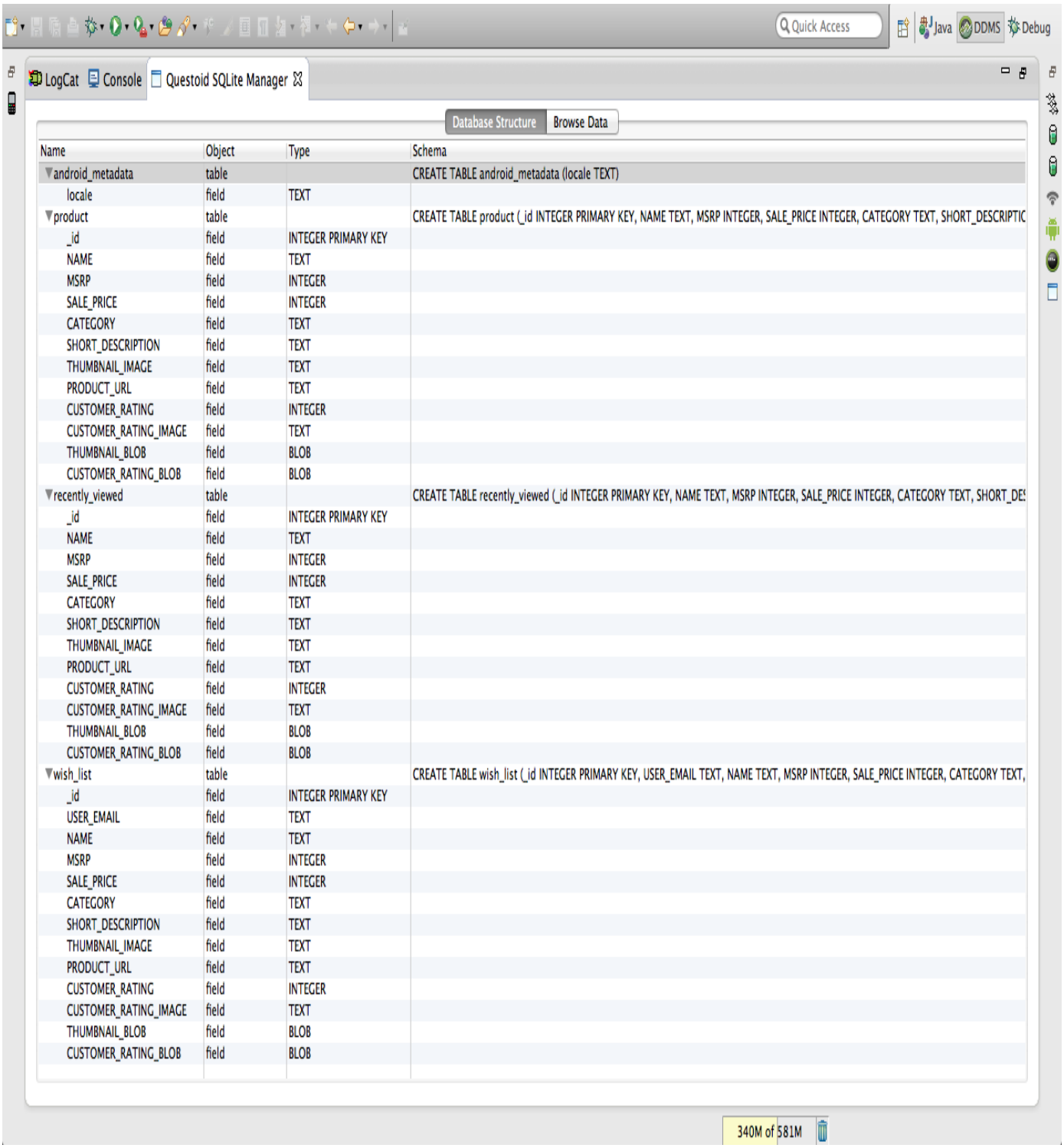
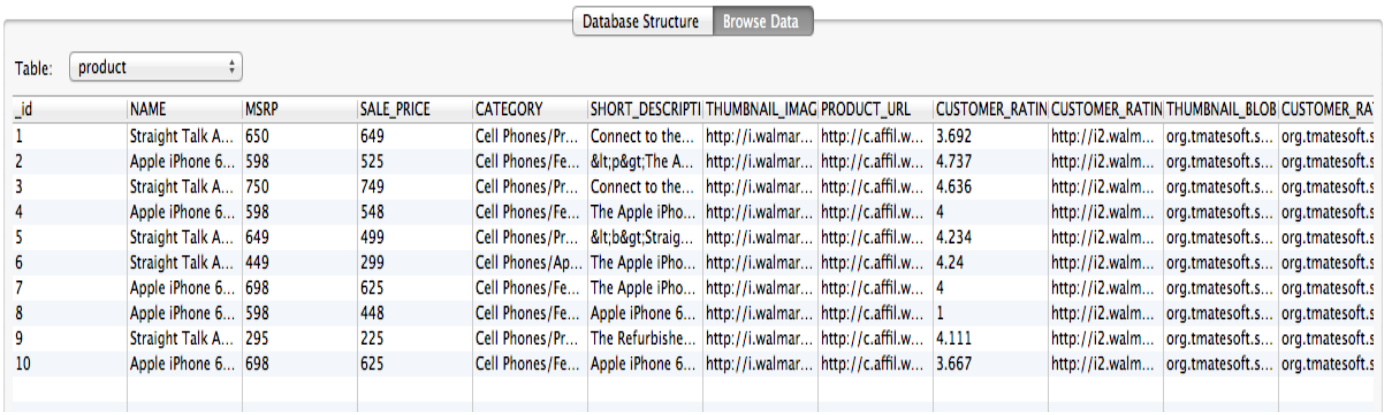


Figure 4.5: Schema for Android App

Product Records for Android App:

The following are the products that were returned for a specific user search.



_id	NAME	MSRP	SALE_PRICE	CATEGORY	SHORT_DESCRIPTI	THUMBNAIL_IMAG	PRODUCT_URL	CUSTOMER_RATIN	CUSTOMER_RATIN	THUMBNAIL_BLOB	CUSTOMER_RA
1	Straight Talk A...	650	649	Cell Phones/Pr...	Connect to the...	http://i.walmar...	http://c.affil.w...	3.692	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
2	Apple iPhone 6...	598	525	Cell Phones/Fe...	<p>The A...	http://i.walmar...	http://c.affil.w...	4.737	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
3	Straight Talk A...	750	749	Cell Phones/Pr...	Connect to the...	http://i.walmar...	http://c.affil.w...	4.636	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
4	Apple iPhone 6...	598	548	Cell Phones/Fe...	The Apple iPho...	http://i.walmar...	http://c.affil.w...	4	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
5	Straight Talk A...	649	499	Cell Phones/Pr...	<b>Straig...	http://i.walmar...	http://c.affil.w...	4.234	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
6	Straight Talk A...	449	299	Cell Phones/Ap...	The Apple iPho...	http://i.walmar...	http://c.affil.w...	4.24	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
7	Apple iPhone 6...	698	625	Cell Phones/Fe...	The Apple iPho...	http://i.walmar...	http://c.affil.w...	4	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
8	Apple iPhone 6...	598	448	Cell Phones/Fe...	Apple iPhone 6...	http://i.walmar...	http://c.affil.w...	1	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
9	Straight Talk A...	295	225	Cell Phones/Pr...	The Refurbishe...	http://i.walmar...	http://c.affil.w...	4.111	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s
10	Apple iPhone 6...	698	625	Cell Phones/Fe...	Apple iPhone 6...	http://i.walmar...	http://c.affil.w...	3.667	http://i2.walm...	org.tmatesoft.s...	org.tmatesoft.s

Figure 4.6: Product Records for Android App

4.3System Interface and Connectivity Design

We have 2 types of interface designs in our system:

1. The middleware system will interact with the external e-commerce API for fetching the item's data, pricing and promotions.
2. The internal connection involves the interface of middleware with Mobile and with UI dashboard and analytics engine.

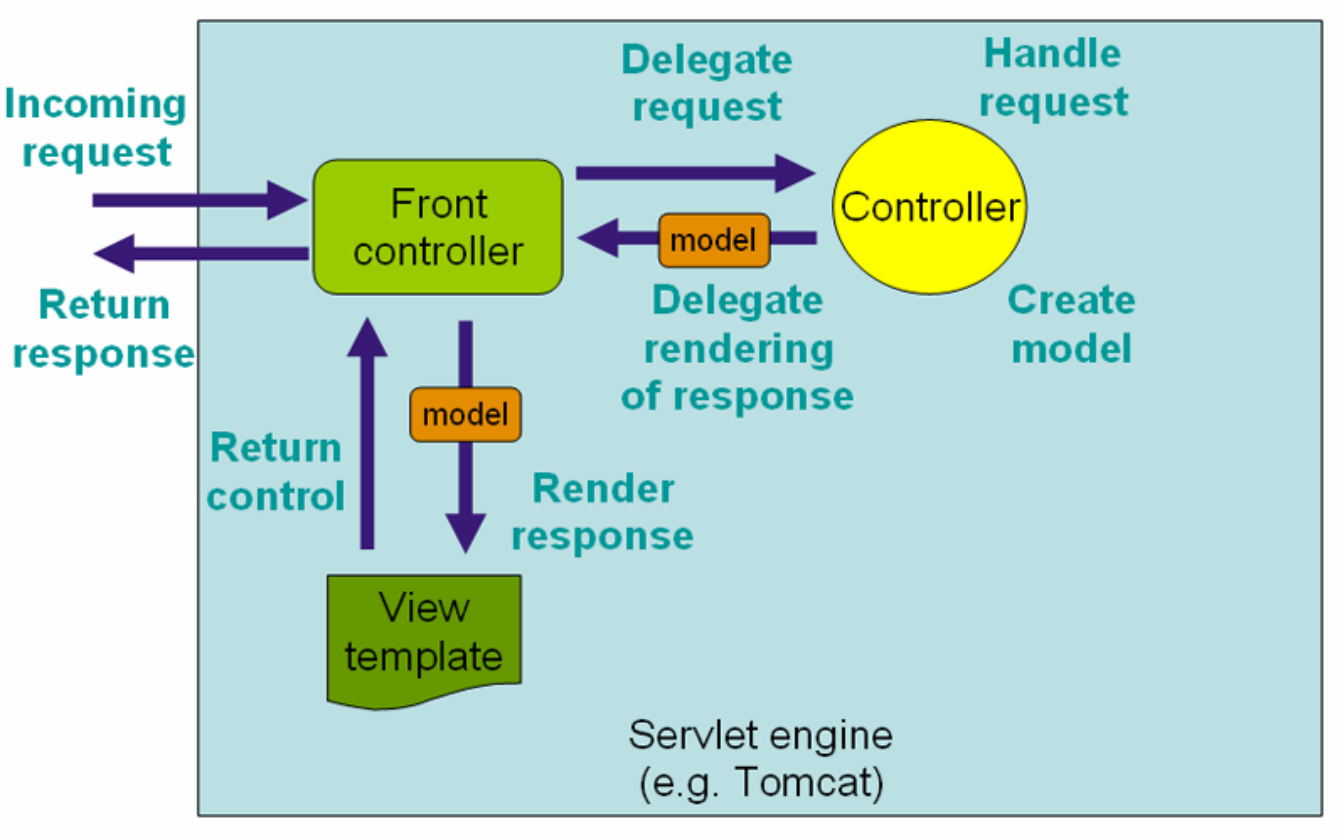


Figure 4.7:Web UI and Dashboard for Business

4.4System User Interface Design

Web UI

AngularJS architecture used for UI along with bootstrap for making the user interface responsive and attractive. Below is the architecture of AngularJs MVC. We have Jsp pages in the views, controllers like product controller, modals, scope comes under the controller section which are defined in the javascript files and objects of the controller are declared in the models.

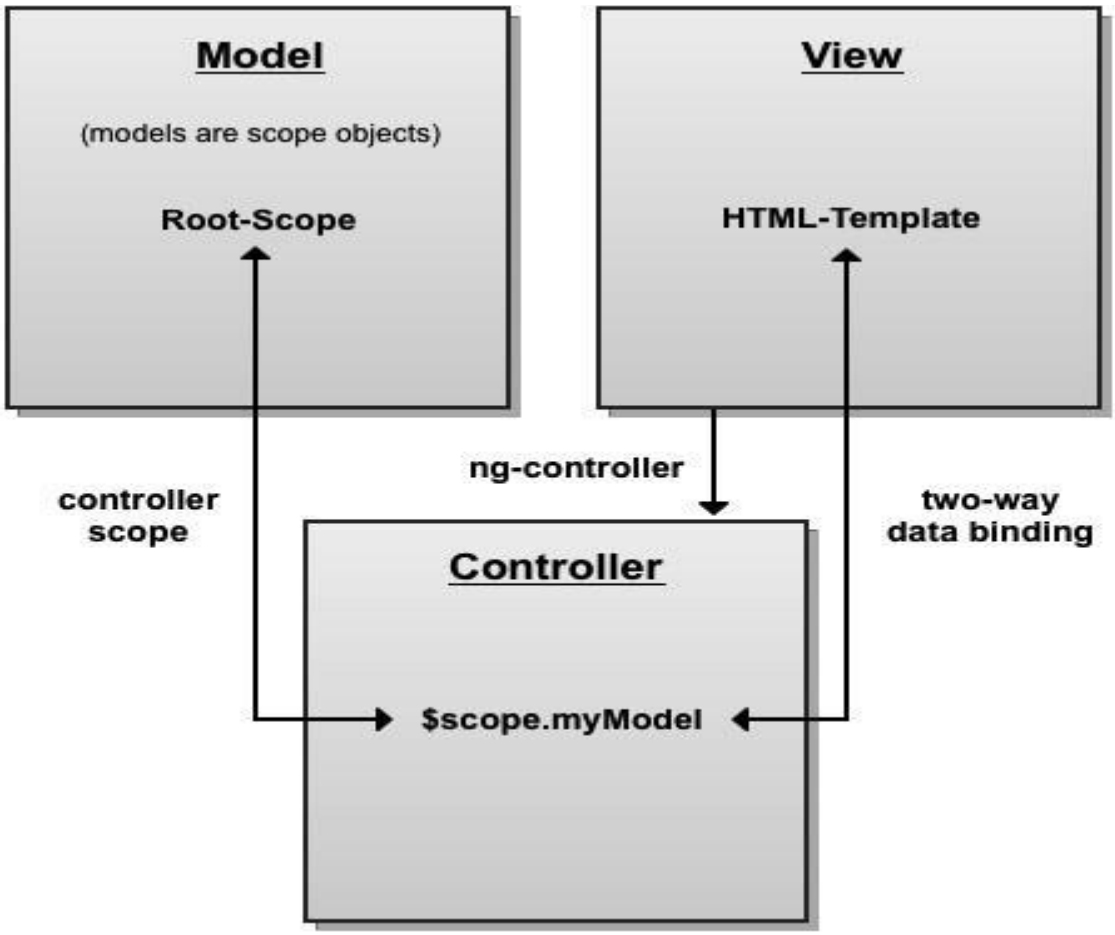


Figure 4.8:MVC Architecture for UI

	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✔ Supported	✔ Supported	N/A	✘ Not Supported	N/A
iOS	✔ Supported	N/A		✘ Not Supported	✔ Supported
Mac OS X	✔ Supported	✔ Supported		✔ Supported	✔ Supported
Windows	✔ Supported	✔ Supported	✔ Supported	✔ Supported	✘ Not Supported

```
app/
----- controllers/
---------- mainController.js
---------- otherController.js
----- directives/
---------- mainDirective.js
---------- otherDirective.js
----- services/
---------- userService.js
---------- itemService.js
----- js/
---------- bootstrap.js
---------- jquery.js
----- app.js
views/
----- mainView.html
----- otherView.html
----- index.html
```

Web UI Screen captures

Home

Home screen of the web application includes signin/signup option, information about the team of developers, attractive user interface and features for the customers to register for the application and sell their product/coupon for free.

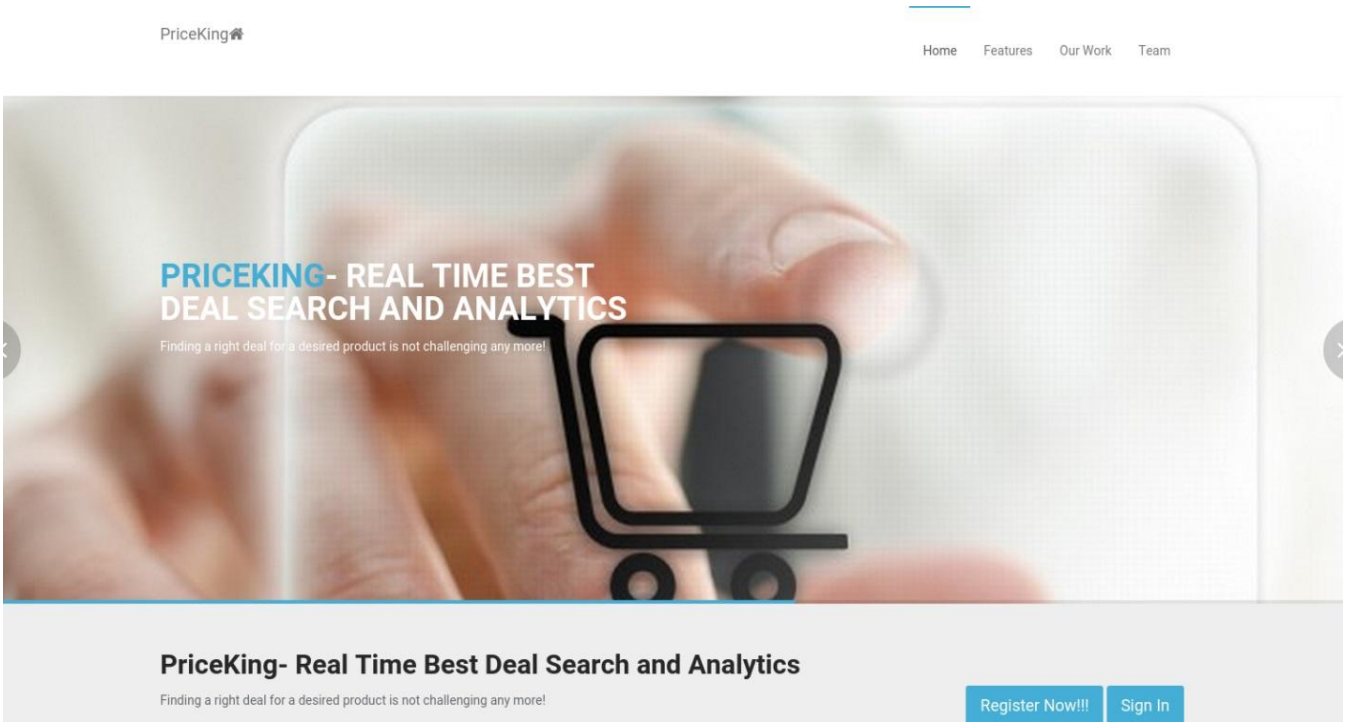


Figure 4.9: Home Screenshot

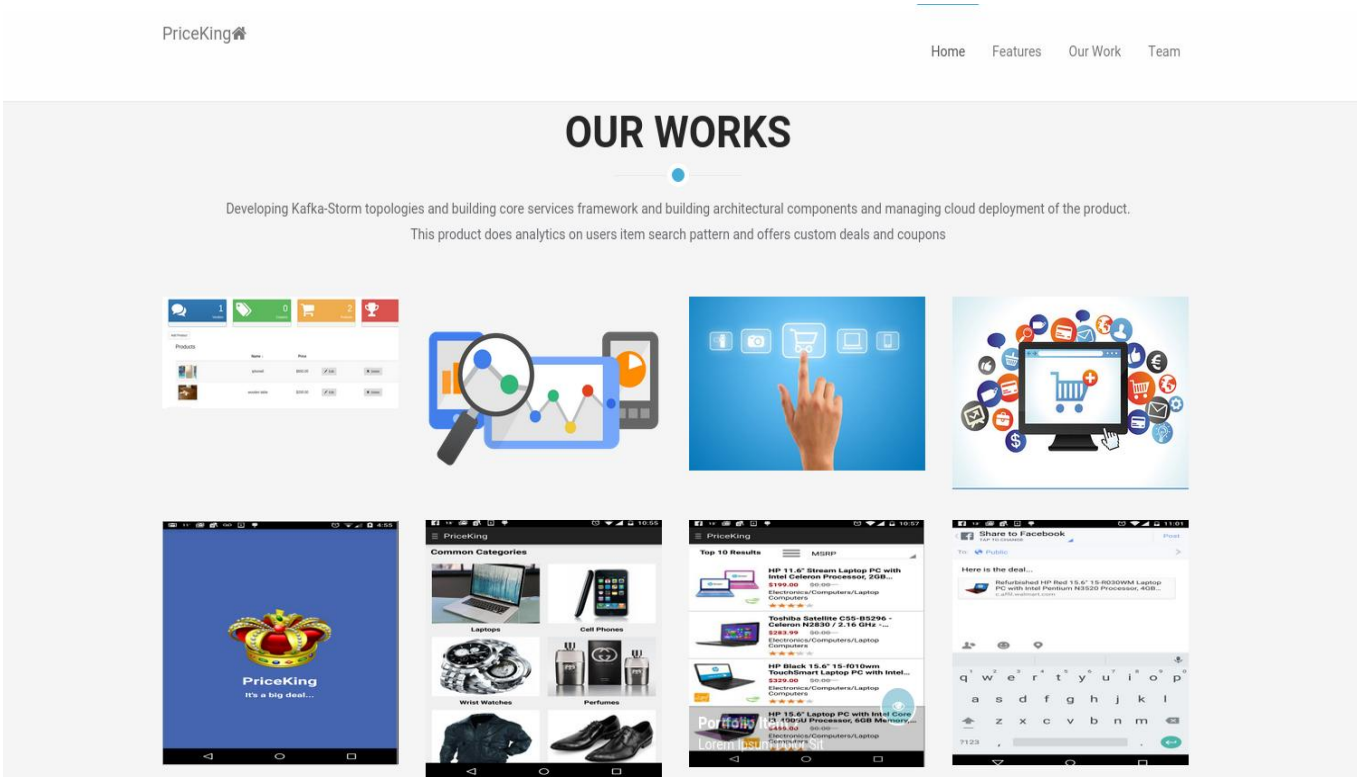


Figure 4.10: Home Screenshot

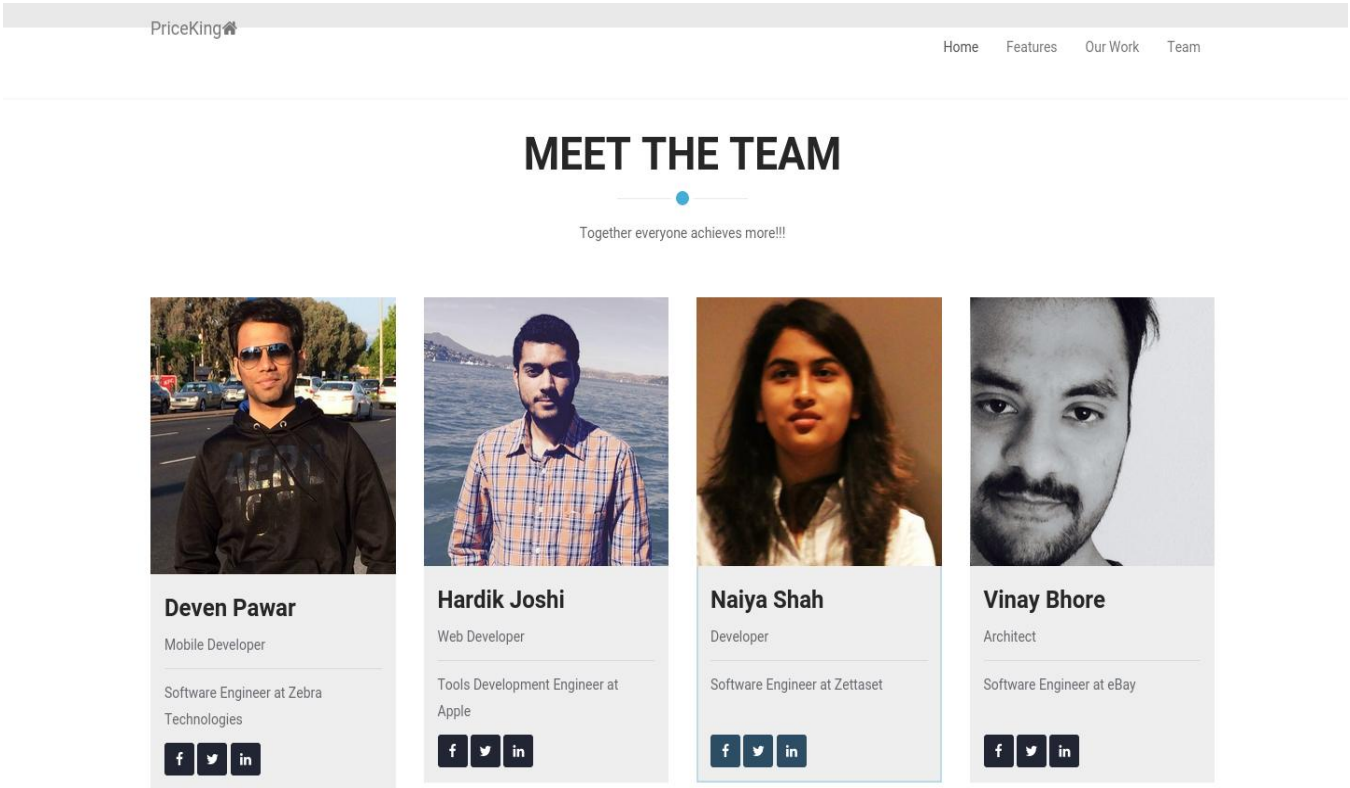


Figure 4.11: Home Screenshot

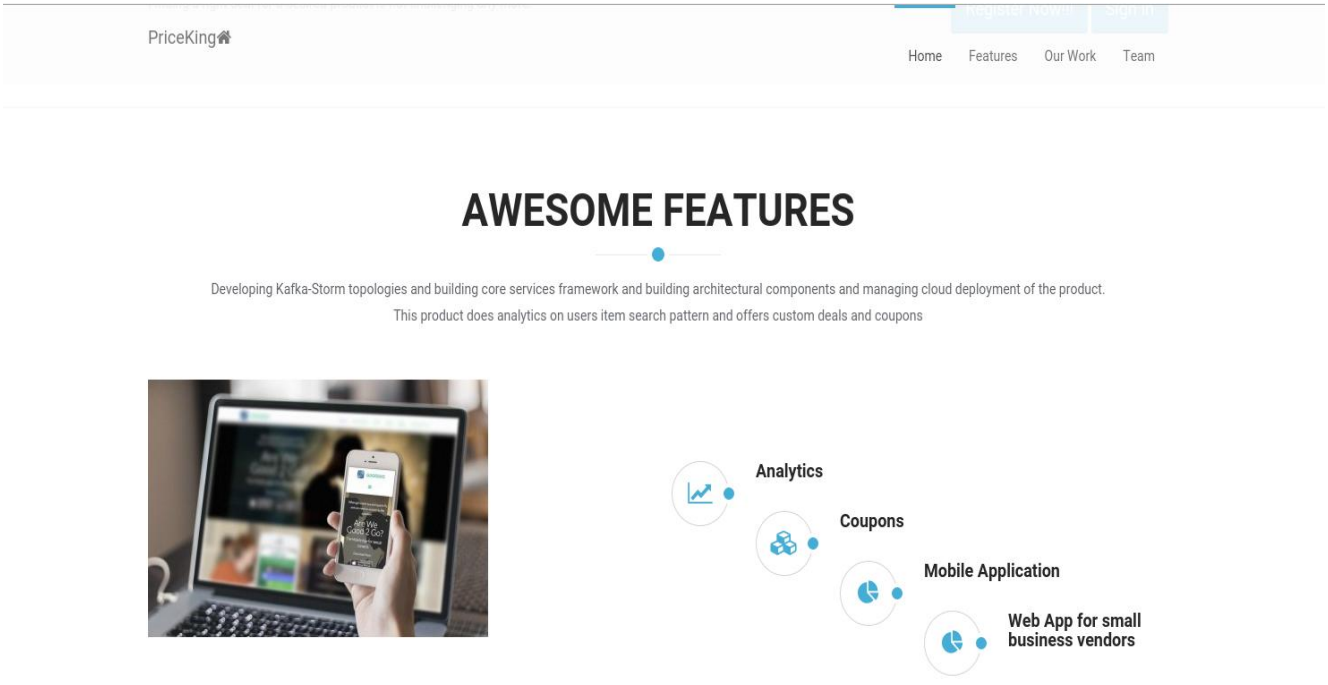


Figure 4.12: Home Screenshot

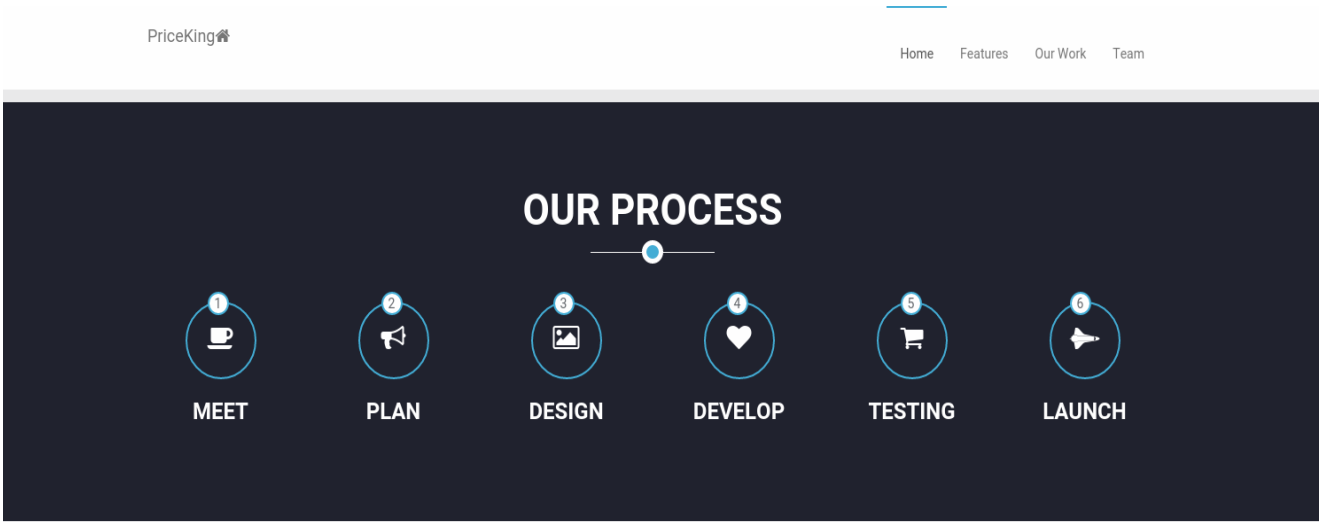


Figure 4.13: Home Screenshot

User Signup and Sign in

User can register by providing the necessary details. All the fields are mandatory. Session is maintained for the active user to avoid inconvenience to the user.

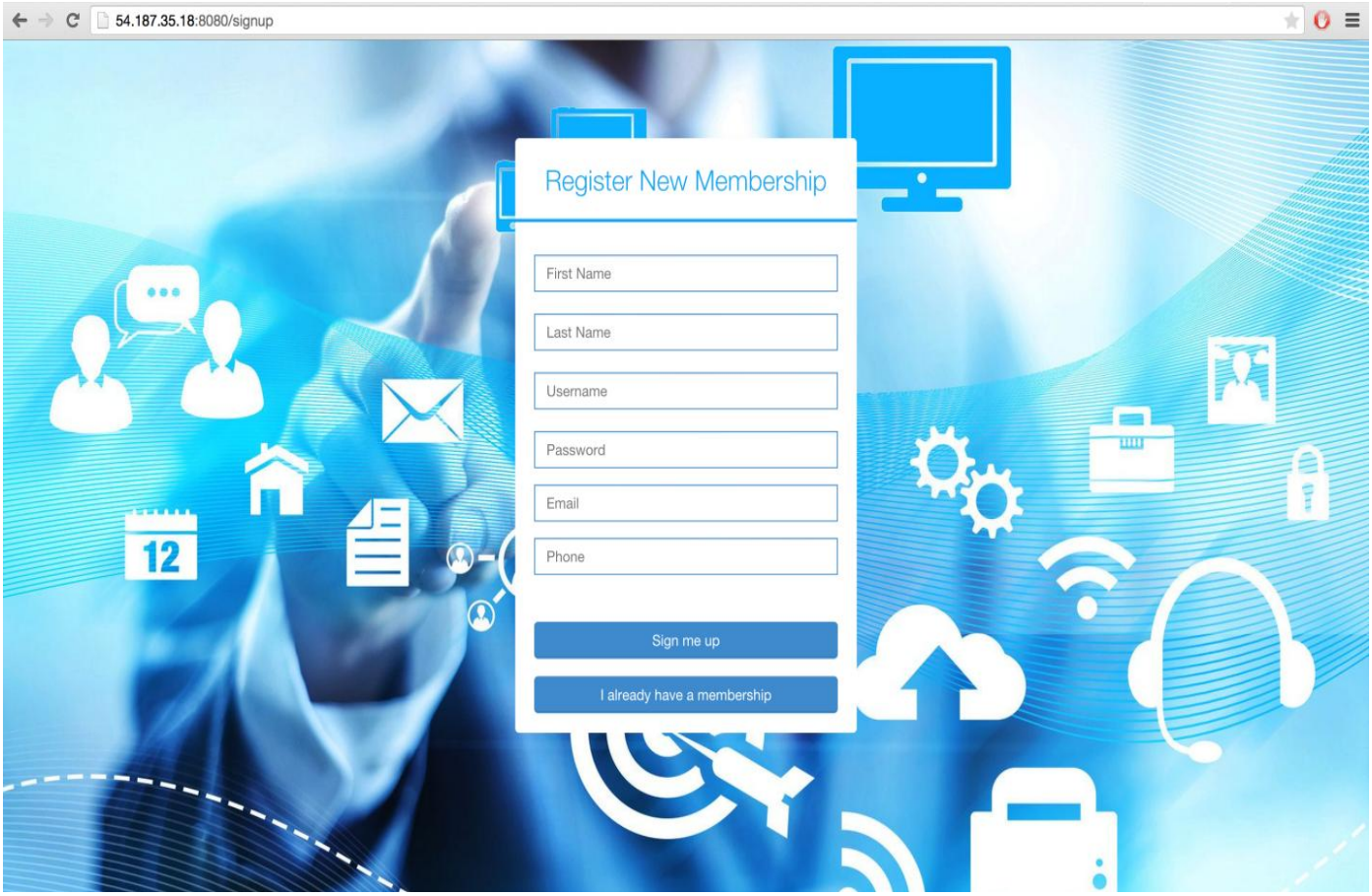


Figure 4.14:Sign Up Screenshot

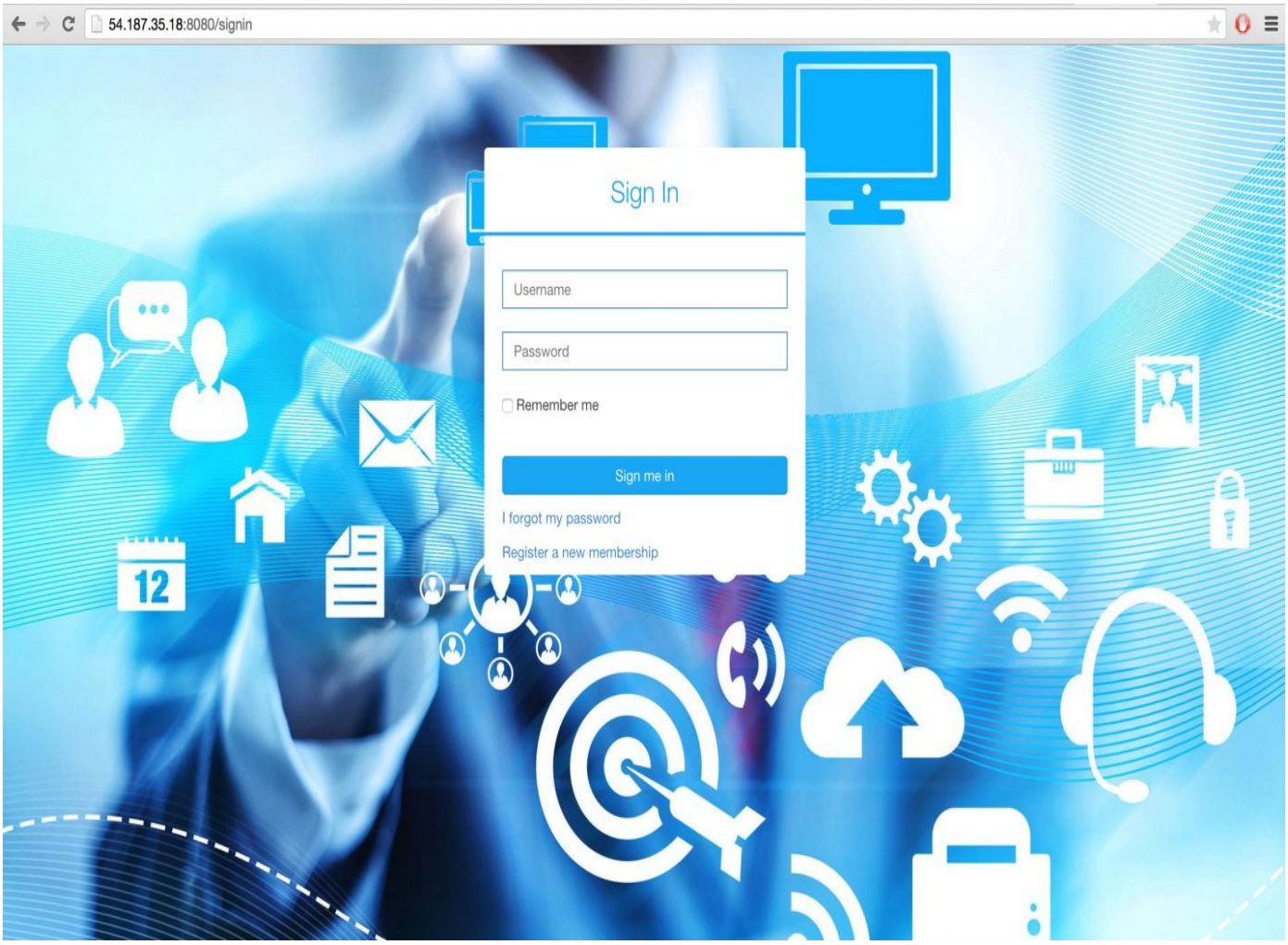


Figure 4.15: SignIn Screenshot

Admin Dashboard

Dashboard includes all the details about users, products, coupons and also analytics. Moreover, it displays total number of users, products and coupons.

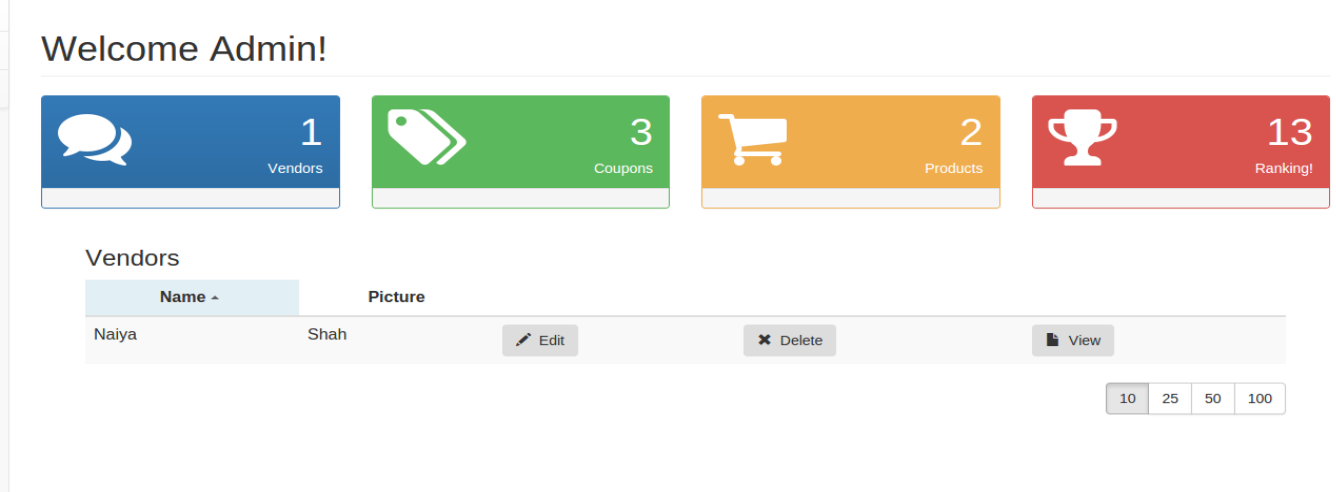
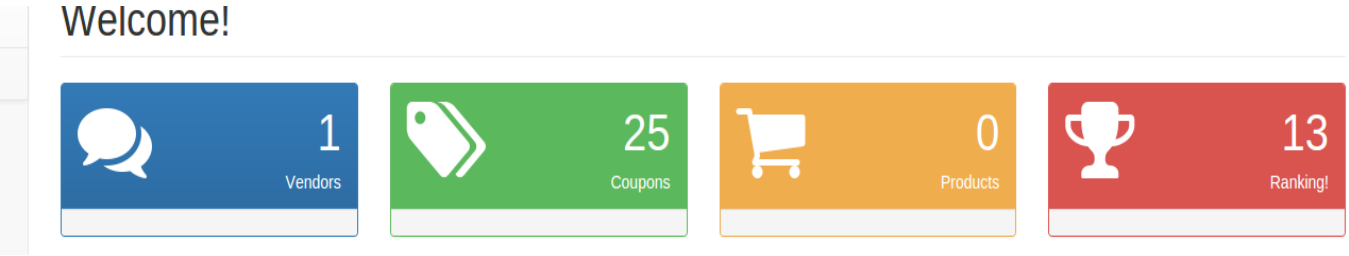


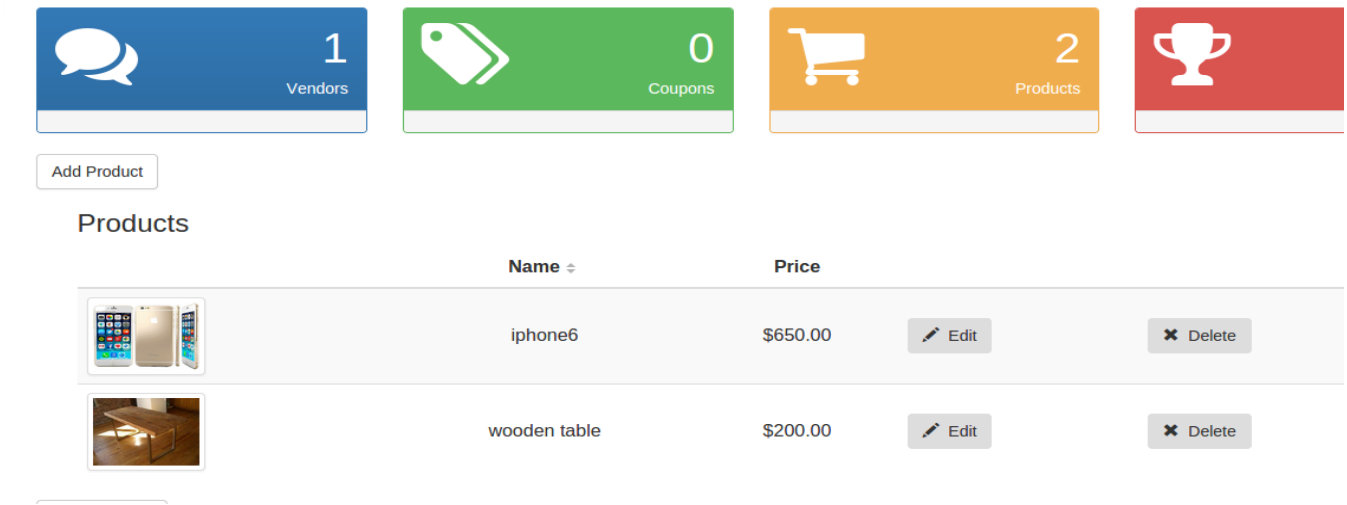
Figure 4.16:AdminDashboard Screenshot

Vendor Dashboard

Dashboard includes all the details about products, coupons created by that particular user and also analytics. It allows user to view/edit/add/delete the products/coupons.Moreover, it displays total number of products and coupons. Below is the panel which displays total number of vendors, total number of coupons uploaded by the particular user, total number of products uploaded by that particular user and ranking based on the number of items uploaded.



Below is the snapshot of the list of the products on the VenderDashboard.



Below is how the list of coupons is displayed on the VendorDashboard

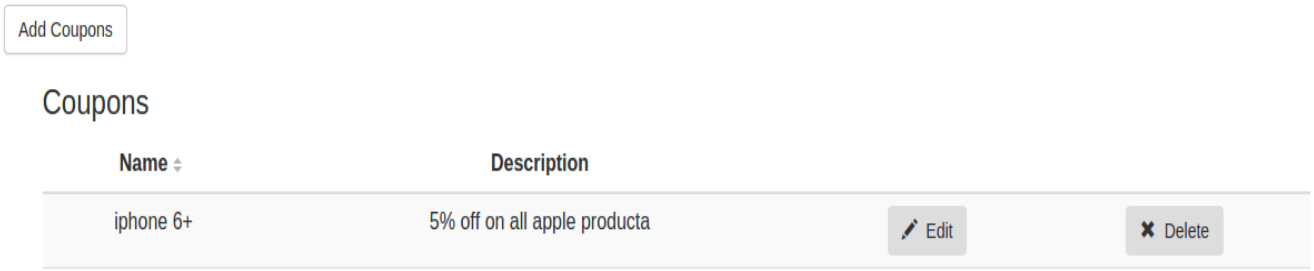
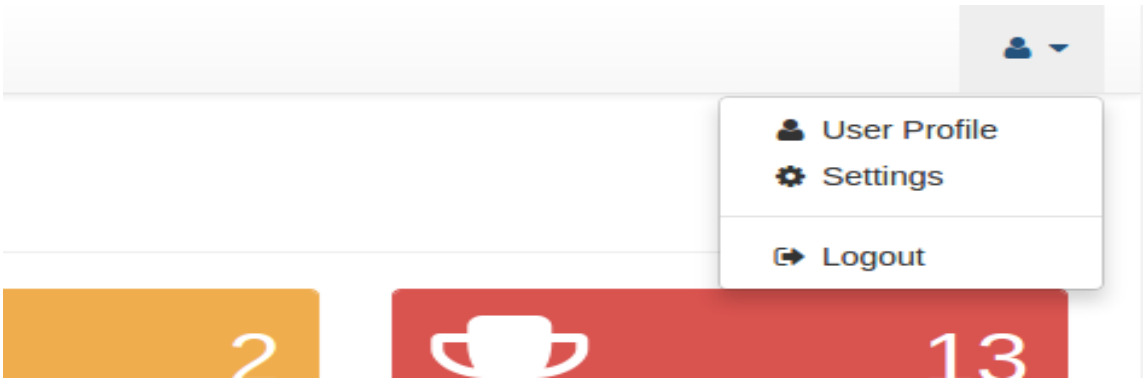


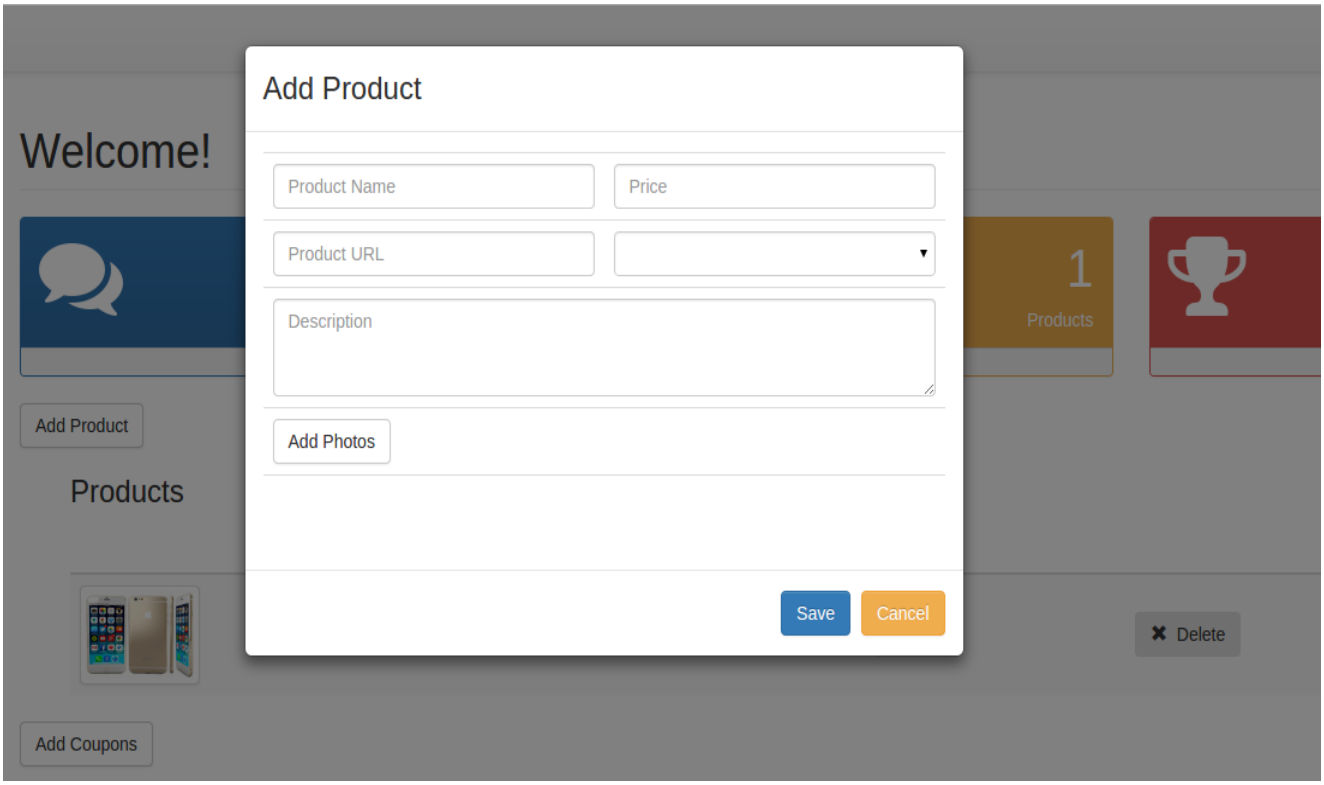
Figure 4.- VendorDashboard Screenshot3

User can edit his profile, logout and change the settings.



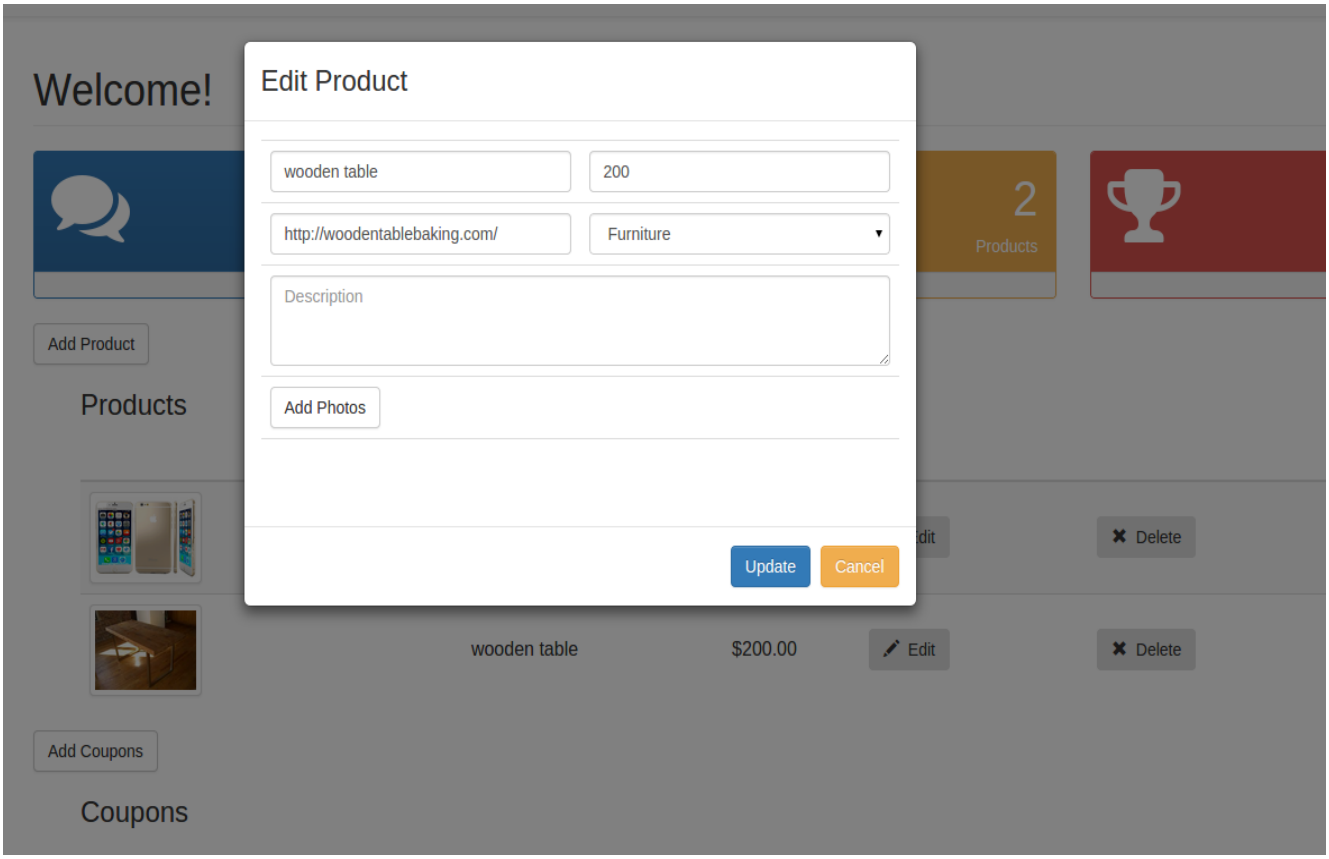
Add Product

User can add products by providing the details like name of the product, description, pictures, category, etc.



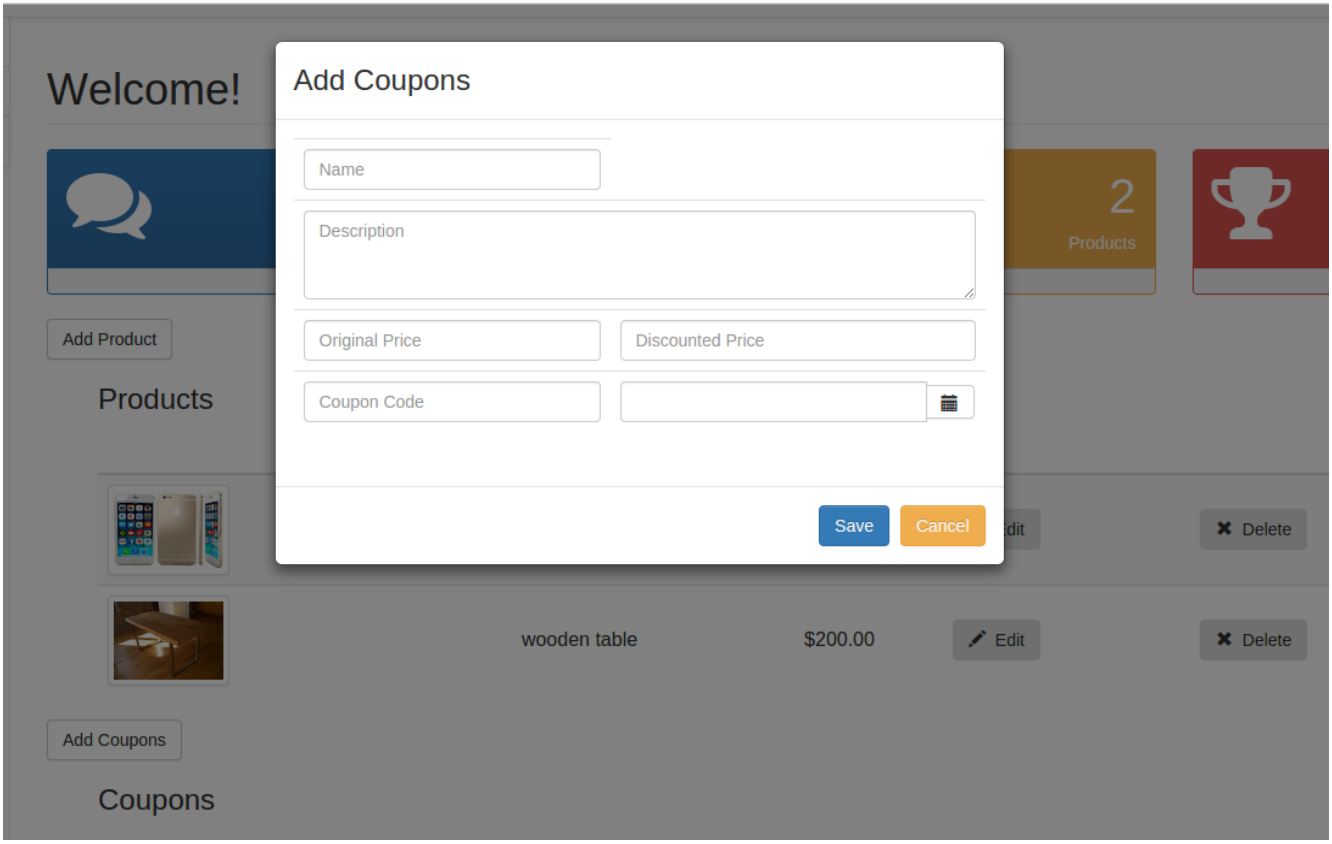
Edit product

User can edit products.

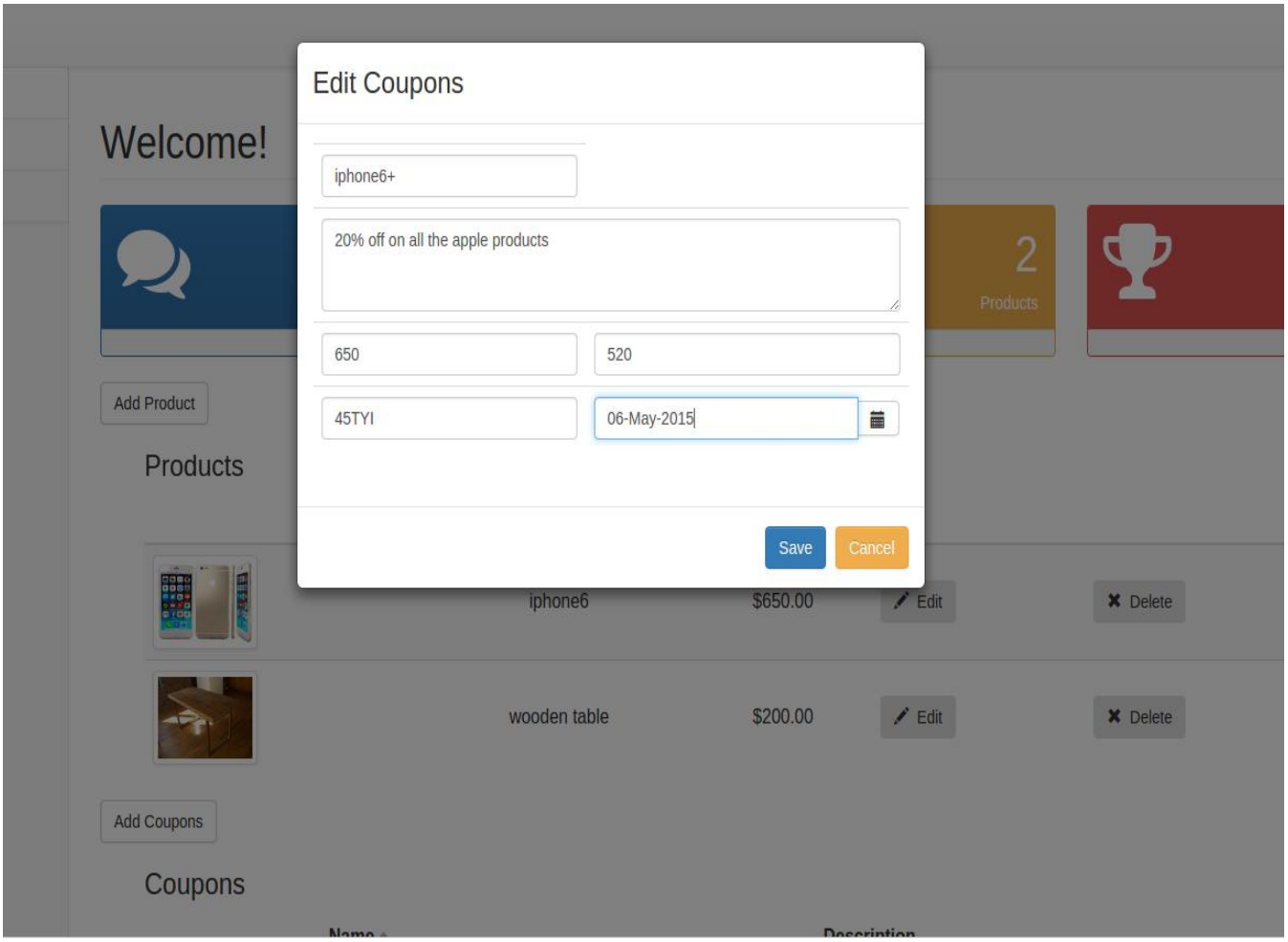


Add Coupon

User can add coupons by providing the details like name of the coupon, description, category, etc.



Edit Coupon
User can edit coupons



Analytics

Users are benefited to view the analytics charts which can help him to improve his business.

Figure 4.- Analytics Screenshot

Mobile UI

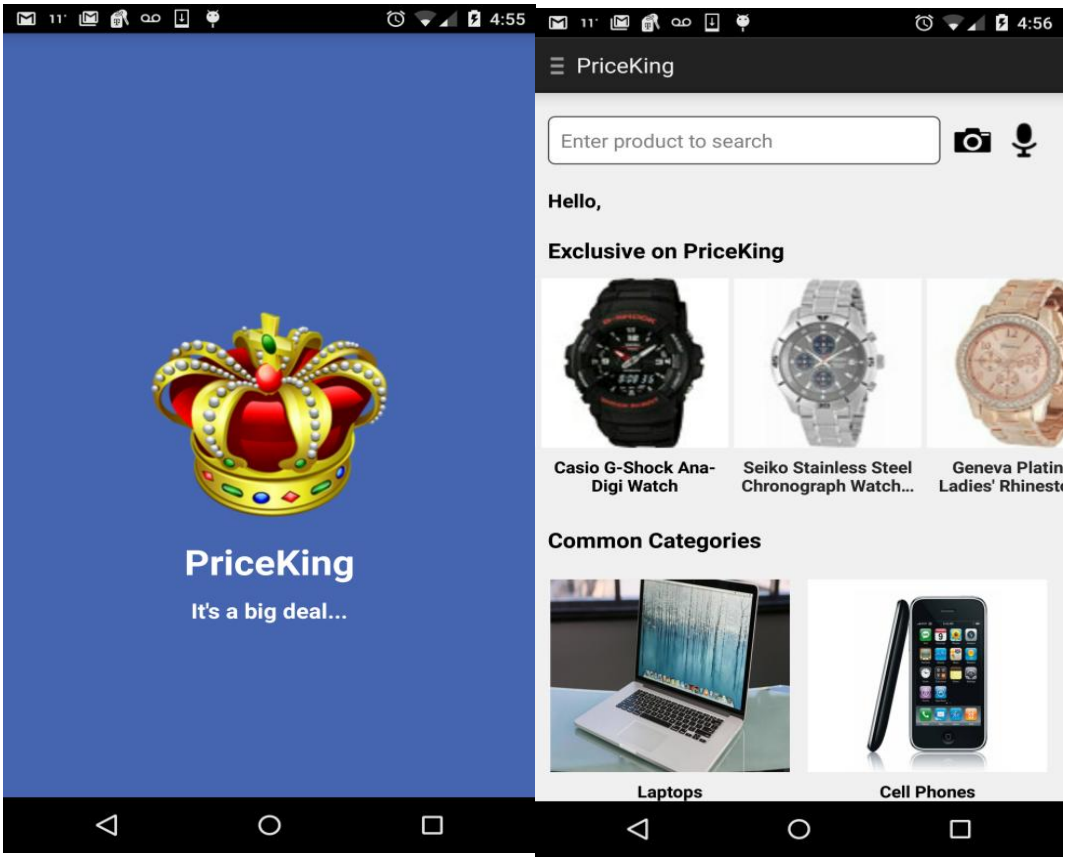
As far as implementation of this application is concerned, we took following requirements in to the account:

SplashScreen with Recommendation Service:

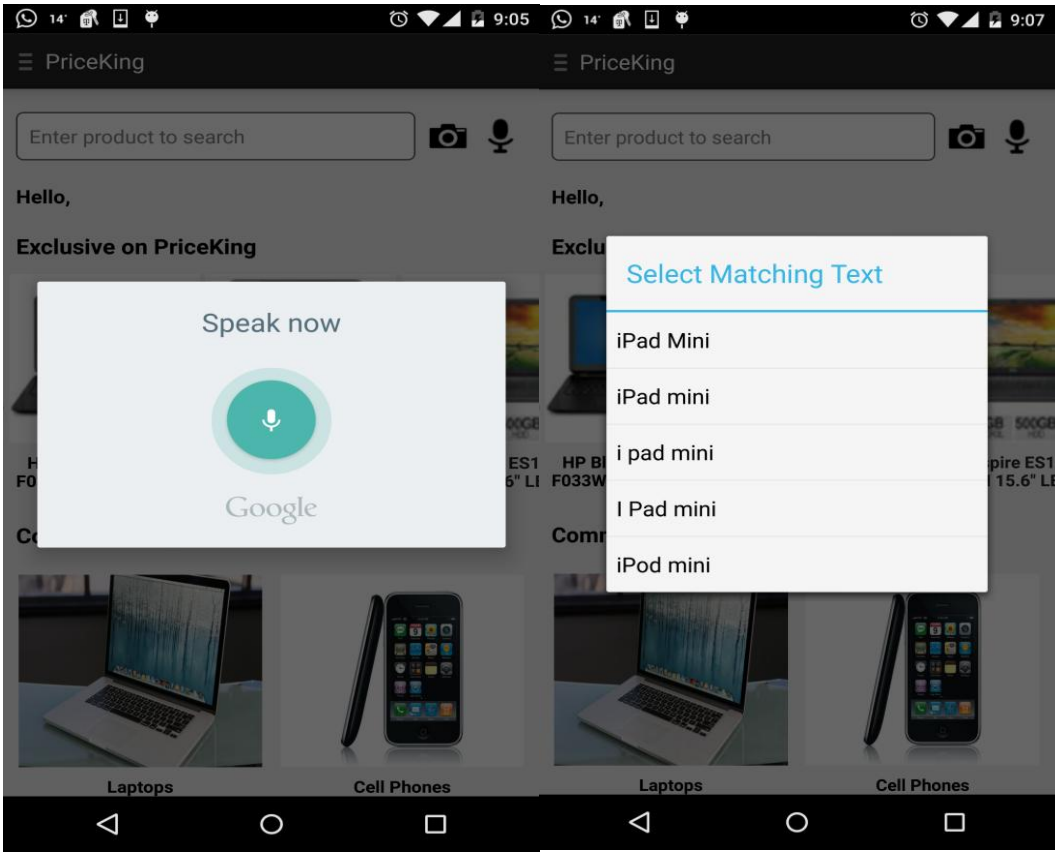
PriceKing application would execute a service to show recommendation of different user interesting products while loading the splashscreen, which is the first screen of the application. It is done so that user doesn't have to wait for extra time for recommendation.

Voice Input for Searching Products:

PriceKing app allows user to provide voice input for user's convenience without letting user to type the search query. It intelligently recognizes the users speech and provides a list of possible outcomes. The user can simply select the matching result and a search service would be fired to provide a list of product. It is as simple as that for a user.



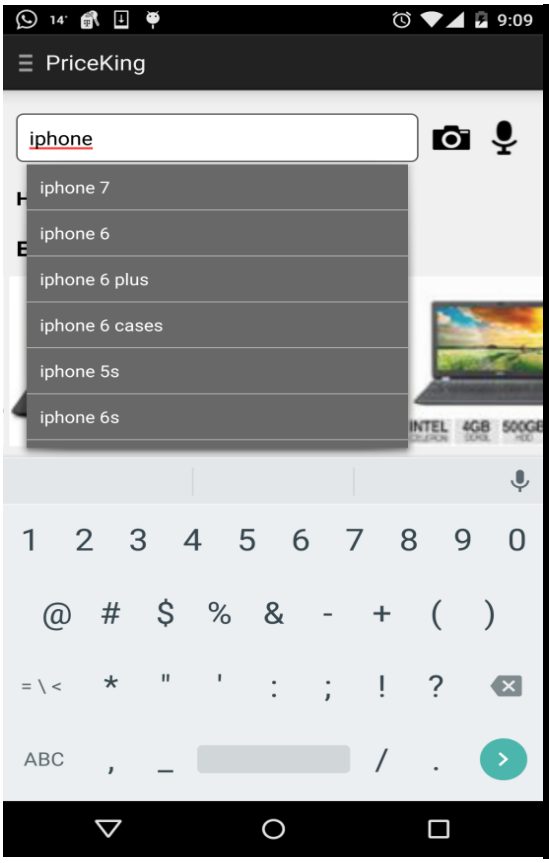
SplashScreen with Recommendation Service Screenshot

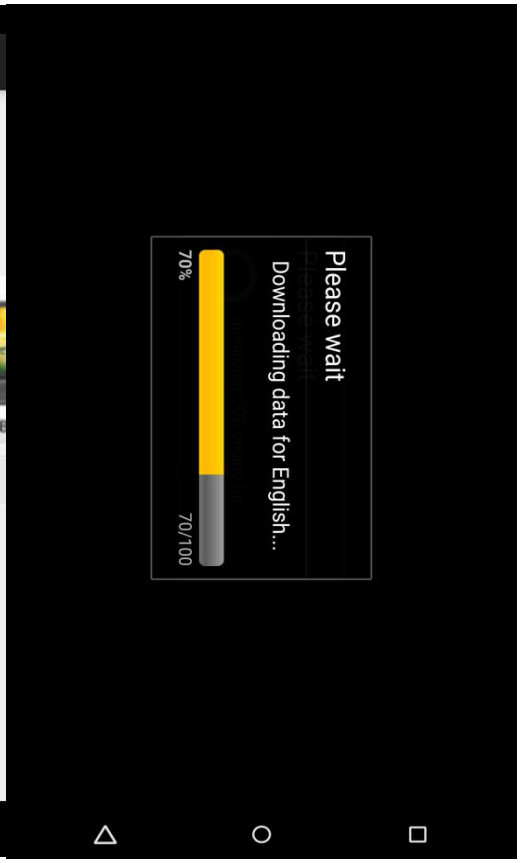


Voice Input for Searching Products Screenshot

Type Ahead Search:

The app allows suggestions for the user while typing a product name or search query. It uses Bing's type ahead API, that would ease typing more keys by providing matching suggestions in the form of list, that the user might want to search for.





Type ahead Screenshot

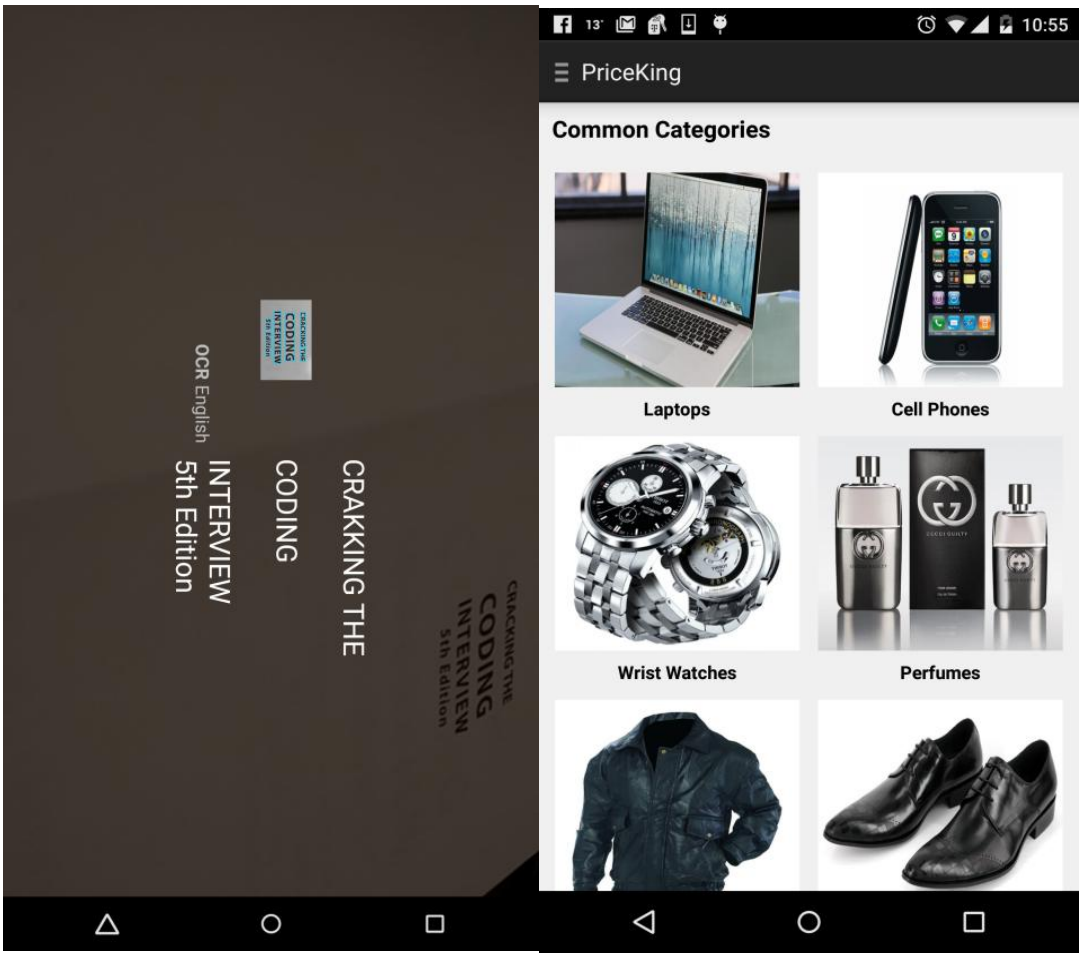
OCR Search Screenshot

OCR Search:

PriceKing provides another mode of providing user input for product, which is Optical Character Recognition (OCR) mode. This opens the camera of your device and allows you to focus on a product that you might see and want to search its price online for a better deal. The app with the help of OCR library captures the text on the product, whose image has been taken and hits a service call with that product name as a query. This is a very handy feature while you are roaming around and see anything that you might want to buy with a good deal.

Common Category Search:

The app allows user to search products from common categories such as Laptops, watches, clothing, shoes, perfumes etc. that provides a list of products based on user selected category.



Common Category Search Screenshot

Product List

Once the user provides search input, the app makes an HTTP based REST call and loads the best result in a list, which is populated on mobile UI. The user can also change the look of this list by pressing appropriate buttons.

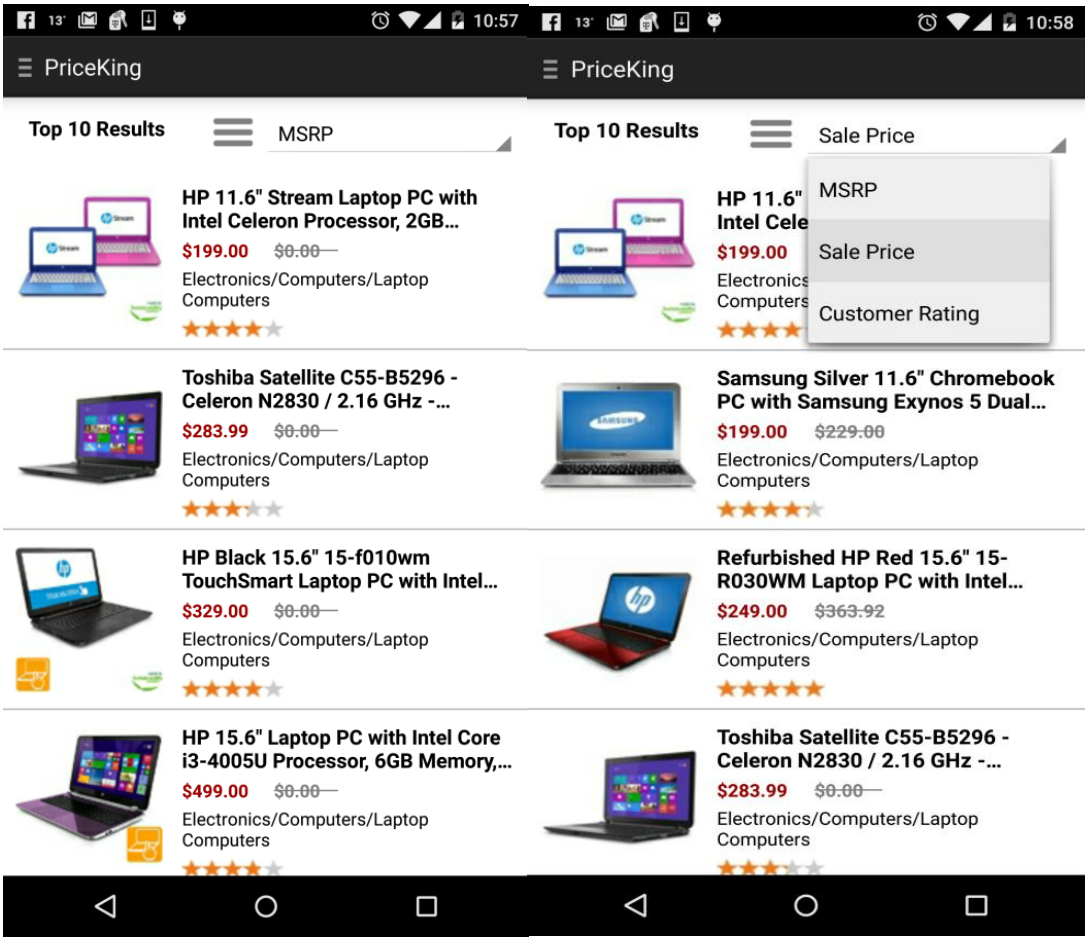
Filter Results:

The app enables user to filter product list based on various parameters for the convenience. Following are the parameters the user can use for filtering:

MSRP – Results would be filtered from lowest MSRP to the highest.

Sale Price: Results would be filtered from lowest sale price to highest.

Customer Rating: Results would be filtered from highest user rating to lowest.



Product List Screenshot Filtered Results Screenshot

Product Detail:

Once the user selects a particular product from the list, its navigated to the detail screen where more details of that product are displayed. These details include, product title, customer rating, thumbnail image of the product, MSRP, Sale Price, discount that user can save, short description about the product and much more.

Share Product Details:

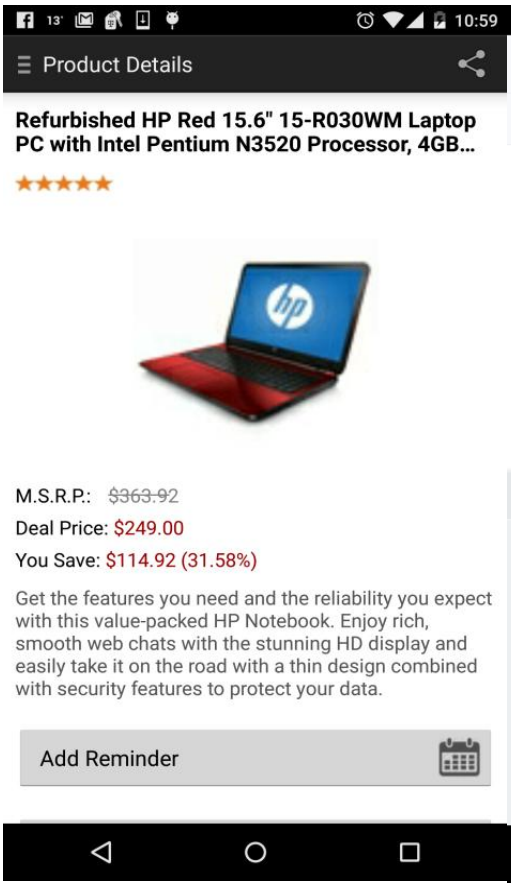
PriceKing allows its users to share any product details on various social networking platforms including Facebook, Twitter, and LinkedIn Email etc. This feature is handy when a user might find a really good deal and want to share it with its friends to promote the deal.

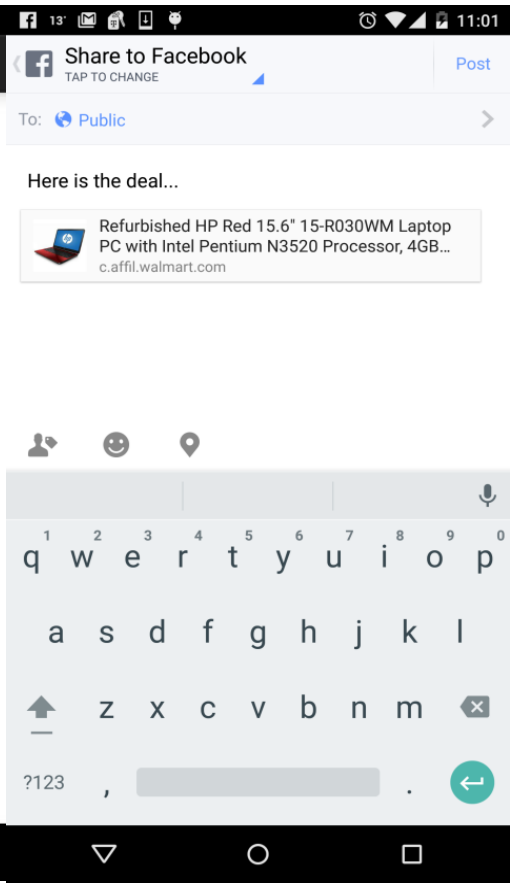
Add Reminder:

The user can add a reminder for any product of interest to view it later when he is busy at that point of time. This reminder would be synced with Calendar and notify user at specified point of time.

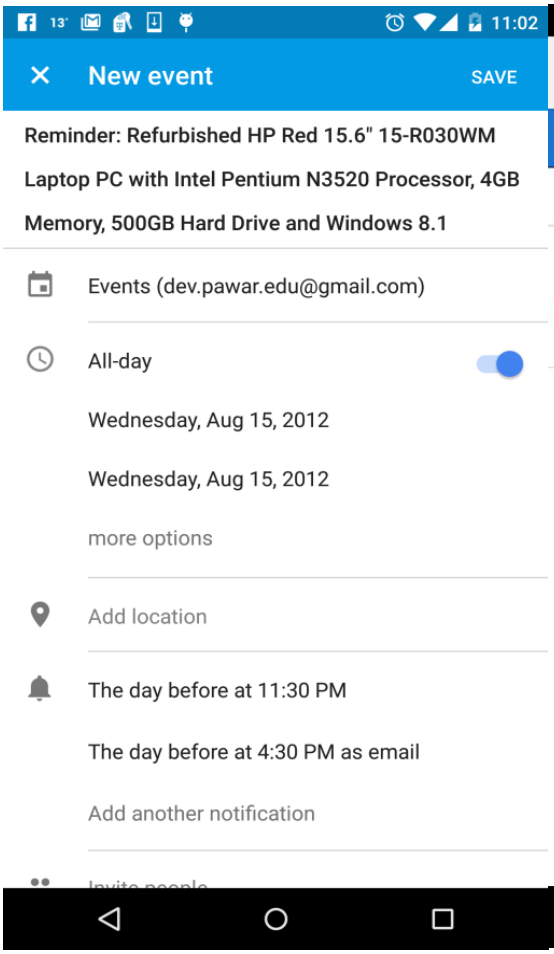
Go to Purchase:

User can visit the actual purchase link through the app if he is satisfied with the product and go ahead to buy it online. The app opens the product URL in a browser to make it easy for the user to buy.





Product details Screenshot Share product details Screenshot

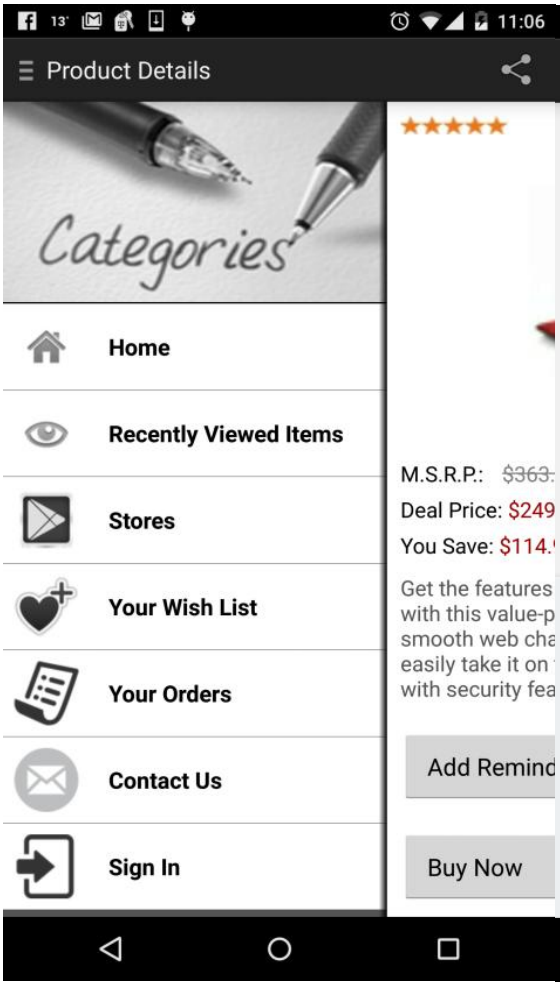


Add Reminder Screenshot

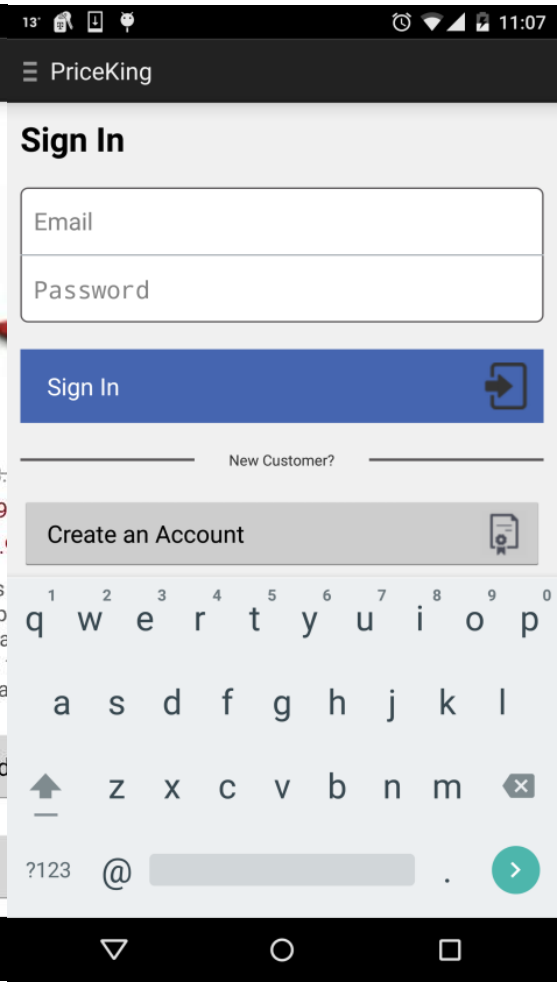
Buy Screenshot

Sign Up and Sign In:

The app allows user to Sign Up and Sign In to avail some of the benefits of the app. Usually any user can download the app and search any product without registering. But at some point of time the user may want to add certain product to its wish list to buy in later time. This time user has to register by providing some information.



Slider Screenshot



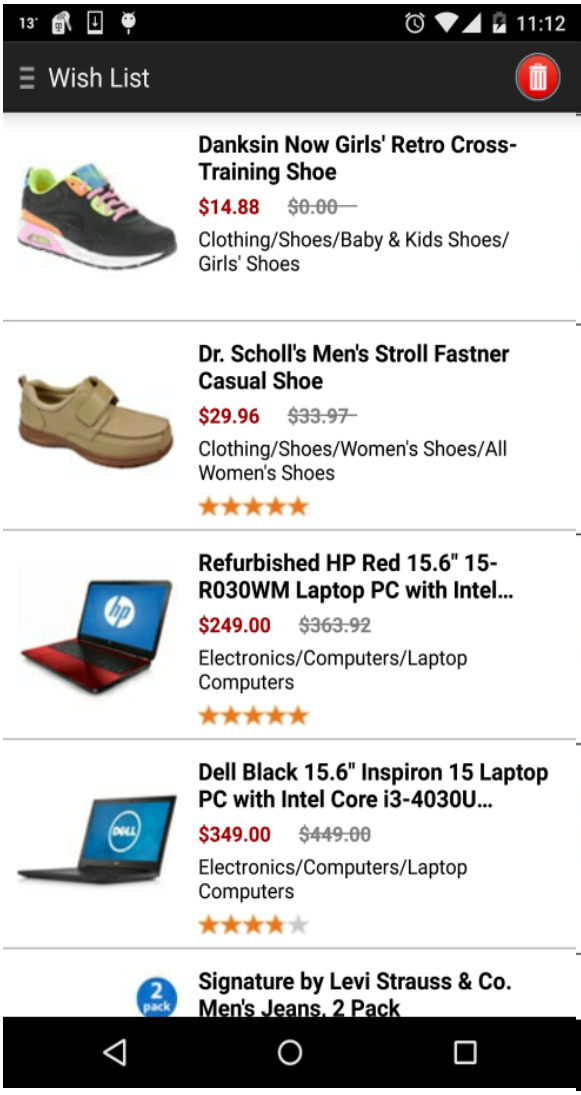
SignIn Screenshot

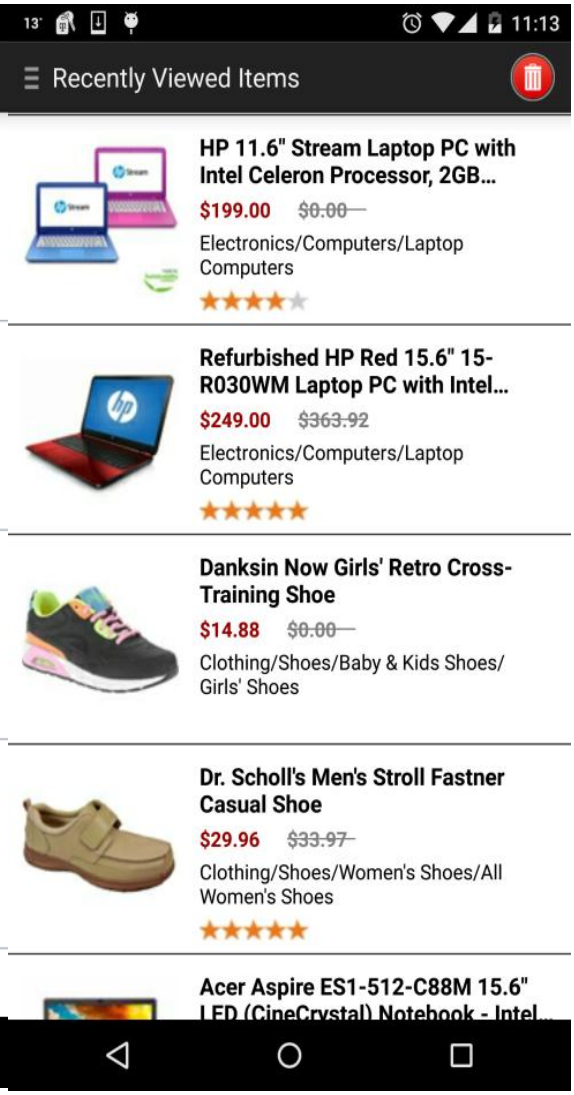
Add to Wish List:

The user can add any product to its wish list that he might want to buy in later point of time. The user can personalize this wish list by dragging items above and below based on priority. The user can also delete some or all the items from the wish list if not required. The condition here is the user must be signed in to add any item to wish list or view any item from the wish list.

Recently Viewed Products:

The app also keeps a track of recently viewed products that may come handy for the user if it wants to see that product details again. Again, the user can delete some or all the recently viewed items in case of overload.



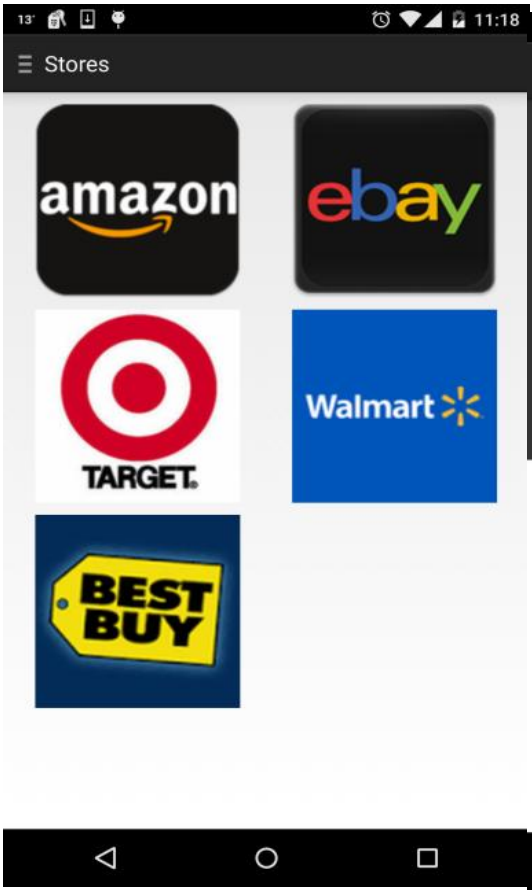


Add to wish list Screenshot

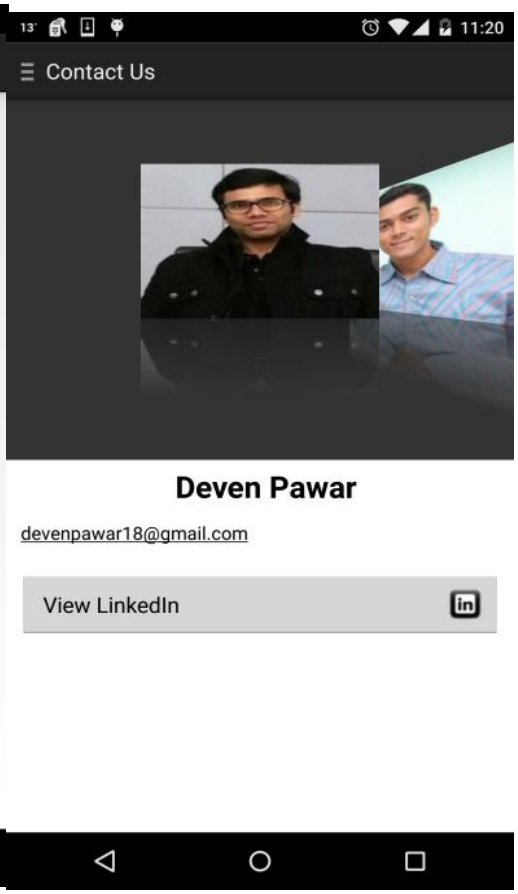
Recently viewed Screenshot

Supported Stores:

This screen would display all the stores form where the products are shown to the user. This list includes some of the top merchants stores such as Amazon, Wal-Mart, eBay, Target and some small businesses that are registered through our web application.



Supported Stores



Contact Us

Contact Us:

The app has a contact us screen that would let user contact us by providing their precious feedback along with the queries. The user can also connect with us through LinkedIn for further assistance.

4.5System Component API and Logic design

The system is implemented as a distributed computing system. The system uses a set of Restful interfaces for communication. These APIs will be used to expose the system to other web clients. It can also be used to pass data among different components of the system. This is the

generalized interface for communication for the whole system. The messages are transferred in the form of JSON objects. The structure of these objects will be as below:

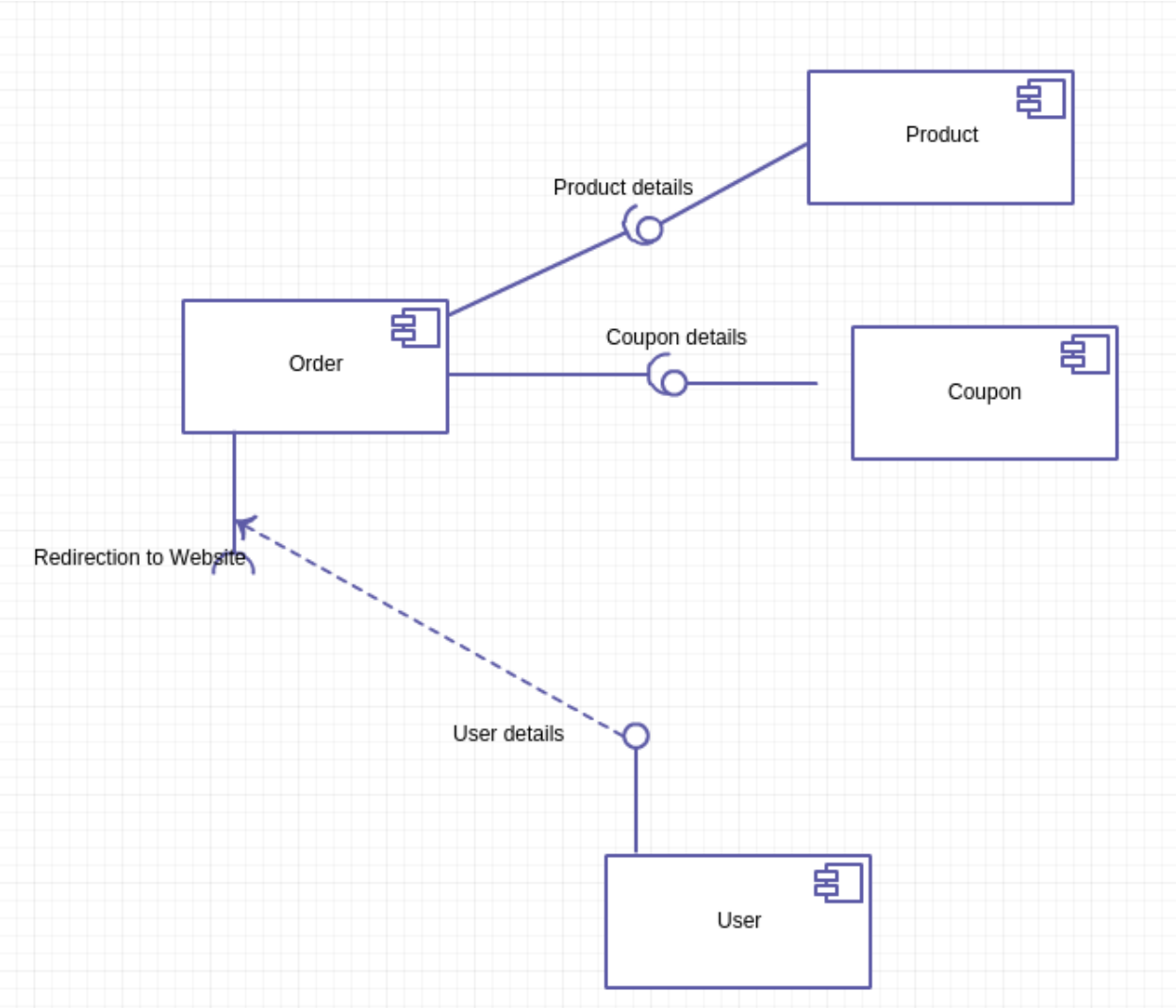


Figure 4.16: Component Diagram

From the above component diagram, it can be seen that there are four major components used at different levels. Some are used at UI level, while others are used at interface, database level. Some components are external which will make the system working.

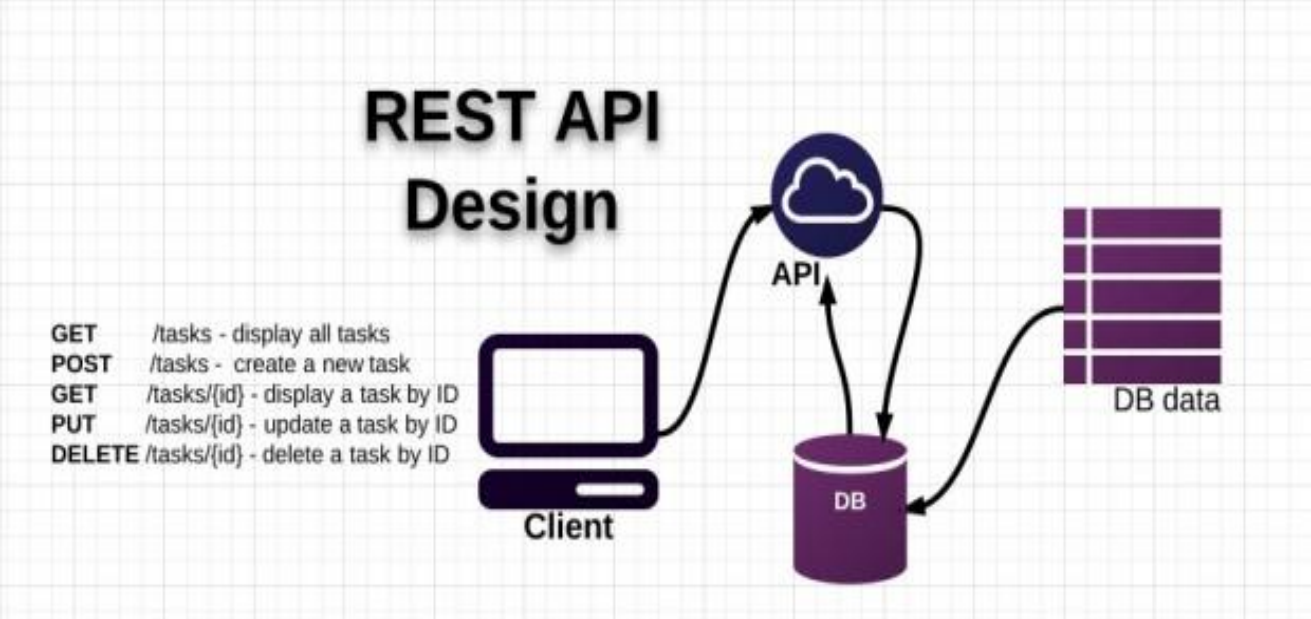


Figure 4.17:Rest API Design

Vendor API:

This API is helpful for storing user related data like username, password, email id and other personal information provided during registration of the system. This API is also responsible for user profiling.

Description	Method	API
New Vendors - Sign Up	POST	/signin
Sign In	POST	/signup
View Vendor	GET	/users
Edit Vendor	PUT	/users/{id}
Delete Vendor	DELETE	/users/{id}

Table 4.1 Vendor API

Product API:

This API is helpful for storing product related data like product name, description, price, category and other details. It is responsible for storing, displaying and editing the product information. Also, for projecting analytics based on the dynamic changes in the product related activities.

Description	Method	API
View Products	GET	/getProducts
Delete Products	DELETE	/product/{id}
Edit Products	PUT	/product/{id}
Save Products	POST	/product

Table 4.2 Product API

Coupon API:

This API is helpful for storing coupon related data like coupon name, description and other details. It is responsible for storing, displaying and editing the coupon information. Also, for projecting analytics based on the dynamic changes in the coupon related activities.

Table 4.3 Coupons API

Description	Method	API
View Coupons	GET	/getCoupons
Delete Coupons	DELETE	/coupon/{id}
Edit Coupons	PUT	/coupon/{id}
Save Coupons	POST	/coupon

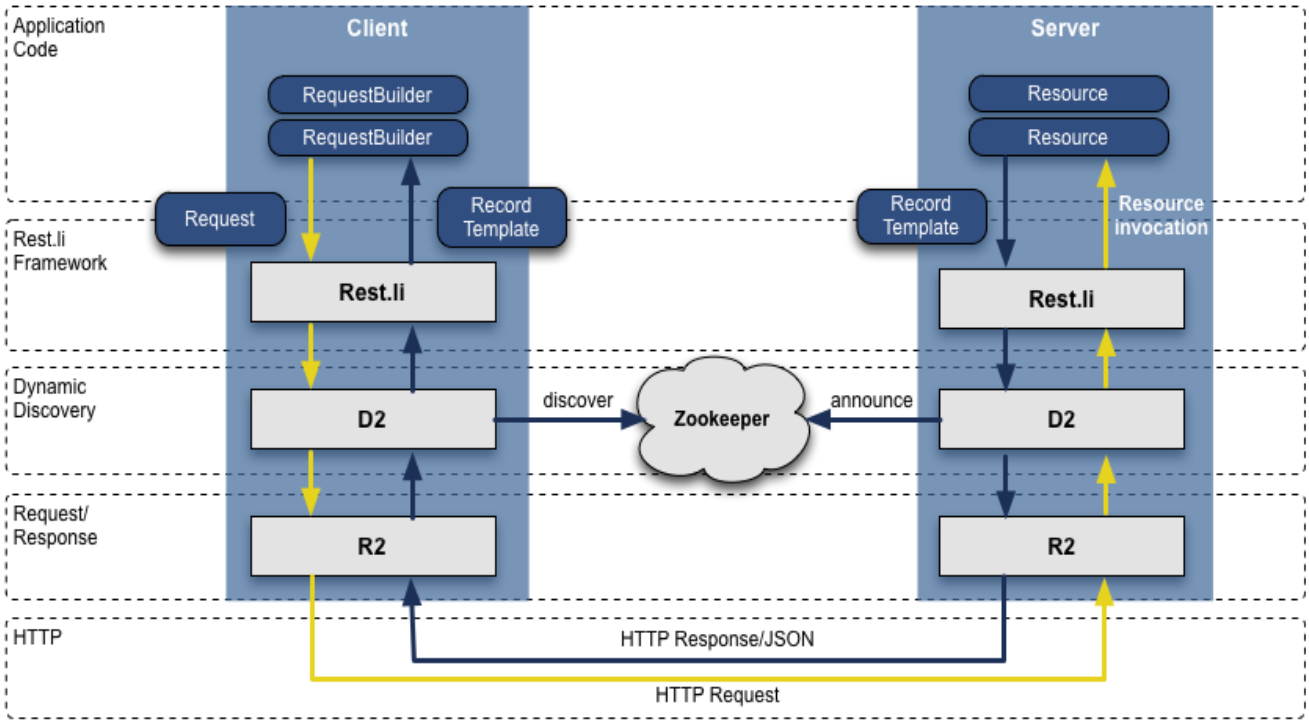


Figure 4.18:Restful Webservices architecture diagram

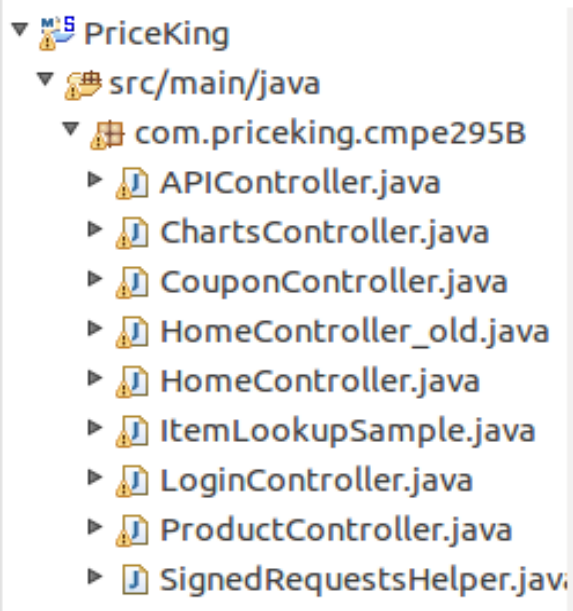
Other API:

This APIs are responsible for displaying the summary to the vendors and admin as well as displaying analytics to vendors.

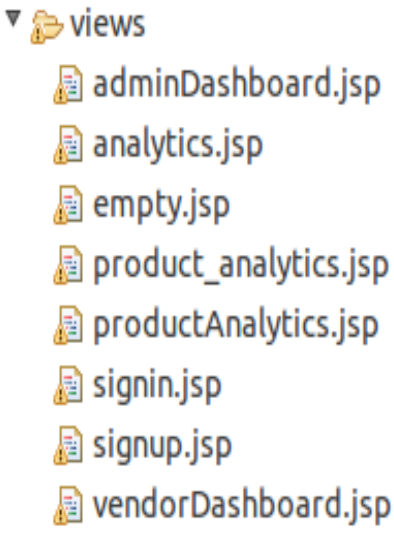
Description	Method	API
Vendor Dashboard	GET	/vendorDashboard
Admin Dashboard	GET	/adminDashboard
Analytics	GET	/analytics

Table 4.4 Other API

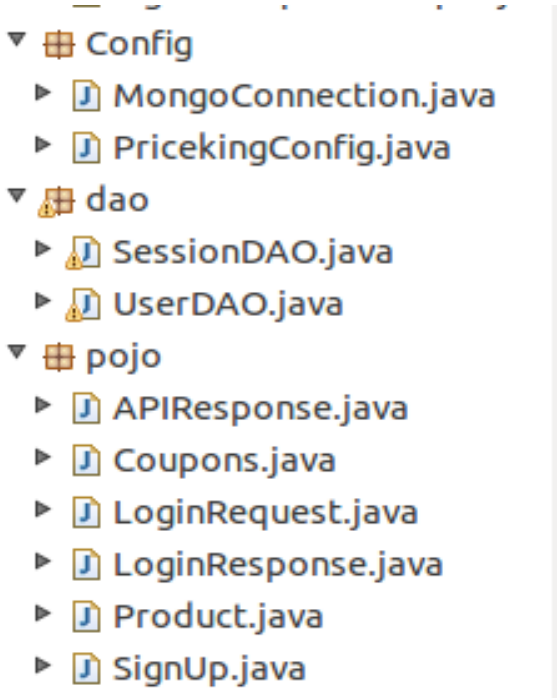
Controller



View



Model



Code snippet

Ebay- The following code is the example of one of the APIs that we have crawled to get the real time data.

```
@RequestMapping(value="/ebay", method=RequestMethod.GET)
public @ResponseBody ArrayList<APIResponse> eBayApiRequest() throws ClientProtocolException, IOException{
    System.out.println("Fetching result from ebay");
    String url = "http://svcs.ebay.com/services/search/FindingService/v1?OPERATION-NAME=findItemsByKeywords&SECURITY-APPNAME=hardikjo-01e9-
    HttpClient client = new DefaultHttpClient();
    HttpGet request = new HttpGet(url);
    request.setHeader("User-Agent", USER_AGENT);
    HttpResponse response = client.execute(request);
    BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));
    ArrayList<APIResponse> apiResponse = new ArrayList<APIResponse>();
    APIResponse ebayResponse;
    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = rd.readLine()) != null) {
        result.append(line);
    }
    JSONObject jObject  = new JSONObject(result.toString());
    JSONArray serverResponse = jObject.getJSONArray("findItemsByKeywordsResponse");
    JSONObject searchArray = serverResponse.getJSONObject(0);
    JSONArray searchResult = searchArray.getJSONArray("searchResult");
    JSONObject itemObject = (JSONObject) searchResult.get(0);
    JSONArray items = itemObject.getJSONArray("item");
    for(int i=0;i<items.length();i++){
        ebayResponse = new APIResponse();
        JSONObject item = items.getJSONObject(i);
        JSONArray sellingStatusArr = item.getJSONArray("sellingStatus");
        JSONObject sellingStatus = sellingStatusArr.getJSONObject(0);
        JSONArray currentPriceArr = sellingStatus.getJSONArray("currentPrice");
        JSONObject currentPriceObj =  currentPriceArr.getJSONObject(0);
        JSONArray primaryCategoryArr = item.getJSONArray("primaryCategory");
        JSONObject primaryCategory = primaryCategoryArr.getJSONObject(0);
        JSONArray categoryNameArr = primaryCategory.getJSONArray("categoryName");
        ebayResponse.setProductName(item.getJSONArray("title").get(0).toString());
        ebayResponse.setPrice(Double.parseDouble(currentPriceObj.getString("__value__")));
        ebayResponse.setProductCategory(categoryNameArr.get(0).toString());
        ebayResponse.setProductDescription("N/A");
        ebayResponse.setThumbnailImage(item.get("galleryURL").toString());
        ebayResponse.setProductUrl(item.get("viewItemURL").toString());
        ebayResponse.setCustomerRating(0);
        apiResponse.add(ebayResponse);
    }
    Gson gson = new Gson();
    return apiResponse;
```

Walmart API code snippet

```java
@RequestMapping(value = "/walmart", method = RequestMethod.GET)
public @ResponseBody ArrayList<APIResponse> walmartApiRequest() throws IllegalStateException, IOException {
    String url = "http://api.walmartlabs.com/v1/search?apiKey=j4x776bmsuzebx8nm4dxbchj&query=iphone6";

    HttpClient client = new DefaultHttpClient();
    HttpGet request = new HttpGet(url);
    ArrayList<APIResponse> apiResponse = new ArrayList<APIResponse>();
    APIResponse walmartResponse;

    // add request header
    request.addHeader("User-Agent", USER_AGENT);

    HttpResponse response = client.execute(request);

    System.out.println("\nSending 'GET' request to URL : " + url);
    System.out.println("Response Code : " +
                   response.getStatusLine().getStatusCode());

    BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = rd.readLine()) != null) {
        result.append(line);
    }

    JSONObject jObject  = new JSONObject(result.toString());
    JSONArray items = jObject.getJSONArray("items");
    for(int i=0;i<items.length();i++){
        walmartResponse = new APIResponse();
        JSONObject item = items.getJSONObject(i);
        System.out.println(item);
        walmartResponse.setProductName(item.get("name").toString());
        walmartResponse.setPrice(Double.parseDouble(item.get("salePrice").toString()));
        walmartResponse.setProductCategory(item.get("categoryPath").toString());
        walmartResponse.setProductDescription(item.get("shortDescription").toString());
        walmartResponse.setThumbnailImage(item.get("thumbnailImage").toString());
        walmartResponse.setProductUrl(item.get("productUrl").toString());
        if(item.has("customerRating")){
        if(item.get("customerRating").toString().isEmpty())
            walmartResponse.setCustomerRating(0);
        else
            walmartResponse.setCustomerRating(Double.parseDouble(item.get("customerRating").toString()));
        }else{
            walmartResponse.setCustomerRating(0);
        }
        apiResponse.add(walmartResponse);

    }
    Gson gson = new Gson();
    System.out.println(gson.toJson(apiResponse));
    System.out.println(result.toString());
    return apiResponse;
}
```

Bestbuy API code snippet

```java
@RequestMapping(value = "/bestbuy", method = RequestMethod.GET)
public @ResponseBody ArrayList<APIResponse> bestBuyApiRequest() throws IllegalStateException, IOException {
    String keyword = "iphone5";
    String url = "http://api.remix.bestbuy.com/v1/products(search="+keyword+")?show=all&format=json&apiKey=x9wtbhpvwfp8kgx86ajzysr3";

    HttpClient client = new DefaultHttpClient();
    HttpGet request = new HttpGet(url);
    ArrayList<APIResponse> apiResponse = new ArrayList<APIResponse>();
    APIResponse walmartResponse;
    request.addHeader("User-Agent", USER_AGENT);
    HttpResponse response = client.execute(request);
    System.out.println("\nSending 'GET' request to URL : " + url);
    System.out.println("Response Code : " +
                   response.getStatusLine().getStatusCode());
    BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = rd.readLine()) != null) {
        result.append(line);
    }
    JSONObject jObject  = new JSONObject(result.toString());
    JSONArray items = jObject.getJSONArray("products");
    System.out.println(items);
    for(int i=0;i<items.length();i++){
        walmartResponse = new APIResponse();
        JSONObject item = items.getJSONObject(i);
        System.out.println(item.get("name"));
        walmartResponse.setProductName(item.get("name").toString());
        walmartResponse.setPrice(Double.parseDouble(item.get("salePrice").toString()));
        JSONArray categoryPath = item.getJSONArray("categoryPath");
        String categorypath="";
        if(categoryPath.length()>0){
            for(int j=0;j<categoryPath.length();j++){
                JSONObject category = categoryPath.getJSONObject(i);
                if(j==categoryPath.length()-1)
                    categorypath+=categorypath + category.get("name");
                else
                    categorypath+=categorypath + category.get("name") +"/";
            }
        }
        walmartResponse.setProductCategory(categorypath);
        walmartResponse.setProductDescription(item.get("shortDescription").toString());
        walmartResponse.setThumbnailImage(item.get("thumbnailImage").toString());
        walmartResponse.setProductUrl(item.get("url").toString());
        if(item.get("customerReviewCount").toString().isEmpty() || "null".equals(item.get("customerReviewCount").toString()))
            walmartResponse.setCustomerRating(0);
        else
            walmartResponse.setCustomerRating(Double.parseDouble(item.get("customerReviewCount").toString()));
        apiResponse.add(walmartResponse);
    }
    Gson gson = new Gson();
    System.out.println(gson.toJson(apiResponse));
    System.out.println(result.toString());
    return apiResponse;
```

compare and sort- Below code is to compare the deals and sort them.

```
@RequestMapping(value = "/compare", method = RequestMethod.GET)
public @ResponseBody ArrayList<APIResponse> comparator() throws IllegalStateException, IOException{
    ArrayList<APIResponse> walmartResponse = walmartApiRequest();
    ArrayList<APIResponse> ebayResponse = eBayApiRequest();
    ArrayList<APIResponse> bestBuyResponse = bestBuyApiRequest();
    ArrayList<ArrayList<APIResponse>> aggregated = new ArrayList<ArrayList<APIResponse>>();
    aggregated.add(walmartResponse);
    aggregated.add(ebayResponse);
    aggregated.add(bestBuyResponse);
    ArrayList<APIResponse> sorted = compareAndSort(aggregated);
    return sorted;
    }

private ArrayList<APIResponse> compareAndSort(
        ArrayList<ArrayList<APIResponse>> aggregated) {
    Gson gson = new Gson();
    ArrayList<APIResponse> listToSort = new ArrayList<APIResponse>();
    System.out.println(gson.toJson(aggregated));
    JSONArray responseArray = new JSONArray(gson.toJson(aggregated));
    for(int i=0;i<responseArray.length();i++){
        JSONArray responseSubArray = responseArray.getJSONArray(i);
        for(int j=0;j<responseSubArray.length();j++){
            JSONObject responseObj = responseSubArray.getJSONObject(j);
            APIResponse response = new APIResponse();
            response.setProductName(responseObj.getString("productName"));
            response.setProductDescription(responseObj.getString("productDescription"));
            response.setProductCategory(responseObj.getString("productCategory"));
            response.setPrice(Double.parseDouble(responseObj.get("price").toString()));
            response.setProductUrl(responseObj.getString("productUrl"));
            response.setThumbnailImage(responseObj.getString("thumbnailImage"));
            response.setCustomerRating(Double.parseDouble(responseObj.get("customerRating").toString()));
            listToSort.add(response);
        }
    }
    Collections.sort(listToSort, new Comparator<APIResponse>() {
          public int compare(APIResponse o1, APIResponse o2) {
              //Sorts by 'price' property
              return o1.getPrice()<o2.getPrice()?-1:o1.getPrice()>o2.getPrice()?1:0;
          }
```

search keywords on api- Finding the product by keyword

```
@RequestMapping(value = "/search", method = RequestMethod.GET)
public @ResponseBody String searchKeywordOnAPI(@RequestParam String userId,@RequestParam String keyword, Locale loca
    BasicDBObject searchObj = new BasicDBObject();
    searchObj.put("username", userId);
    searchObj.put("keyword", keyword);

    UserDAO userDAO = UserDAO.getInstance();

    boolean isUserValid = userDAO.isValidUser(userId);

    //System.out.println(searchObj);
    /*if(isUserValid)
        TestProducer.produceRequest(searchObj);
    else
        return "Invalid Username";*/

    return null;
}
```

api response pojo-Getter Setter methods of the API response fields

```
public class APIResponse  {
    private String productName;
    private String productDescription;
    private String productCategory;
    private double price;
    private String thumbnailImage;
    private double customerRating;
    private String productUrl;
    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    public String getProductDescription() {
        return productDescription;
    }
    public void setProductDescription(String productDescription) {
        this.productDescription = productDescription;
    }
    public String getProductCategory() {
        return productCategory;
    }
    public void setProductCategory(String productCategory) {
        this.productCategory = productCategory;
    }
    public String getThumbnailImage() {
        return thumbnailImage;
    }
    public void setThumbnailImage(String thumbnailImage) {
        this.thumbnailImage = thumbnailImage;
    }
    public String getProductUrl() {
        return productUrl;
    }
    public void setProductUrl(String productUrl) {
        this.productUrl = productUrl;
```

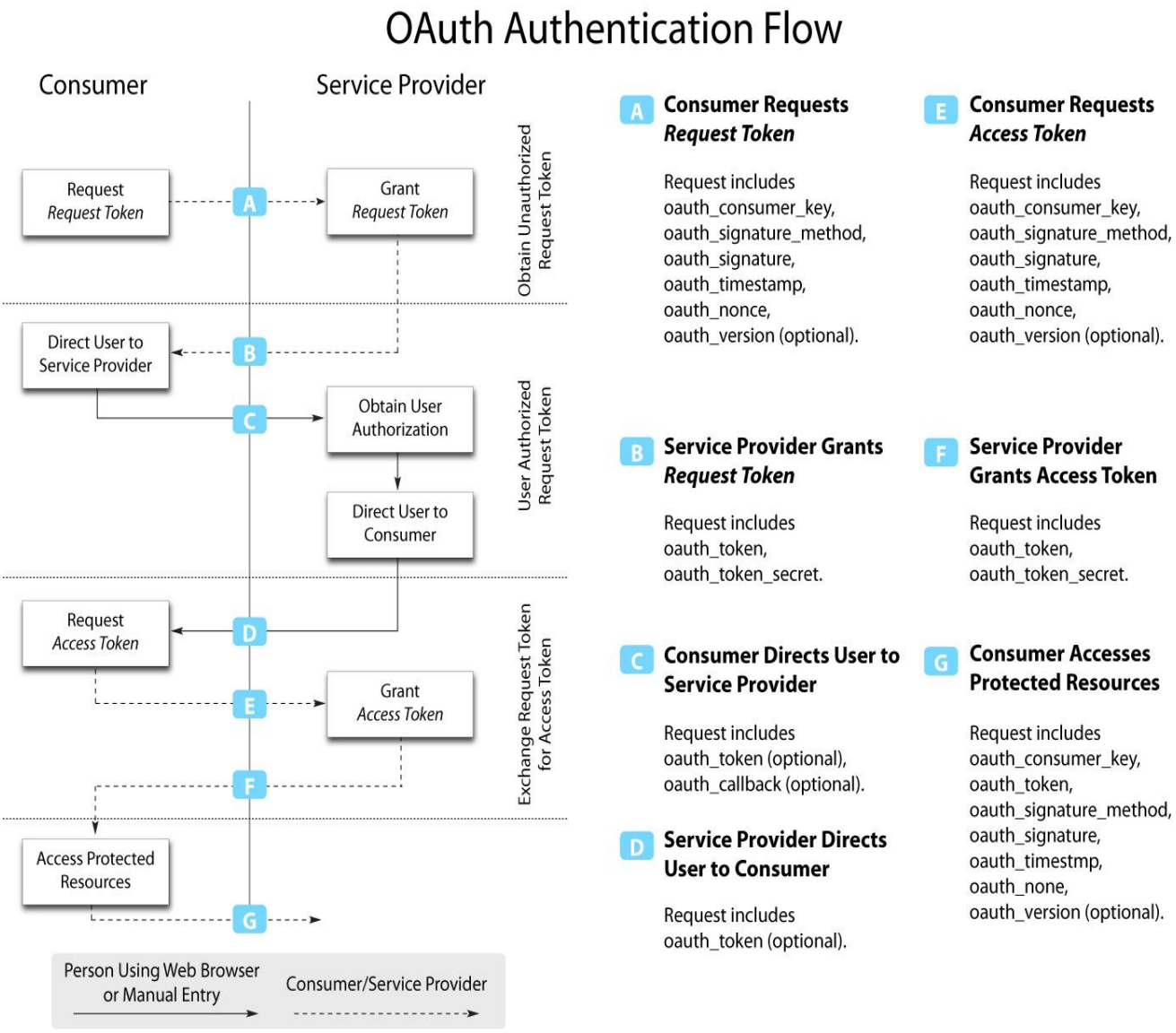


chart controller- Chart controller has the code for analytics that is offered to the small business vendors.

```
@RequestMapping(value = "/analyticsChart", method = RequestMethod.GET)
public @ResponseBody ArrayList<Object>  analyticsChart(Locale locale, Model model) {
    MongoClient mongoClient = MongoConnection.getNewMongoClient();
    ArrayList<Object> response = new ArrayList<Object>();
    DB configDB = mongoClient.getDB("priceking");
    DBCollection productCollection = configDB.getCollection("products");
    DBObject group = new BasicDBObject("$group", new BasicDBObject("_id",new BasicDBObject("productCategory", "$productCategory")).append("count", new BasicDBObject("$
    HashMap<String,Integer> productMap = new HashMap<String,Integer>();
    HashMap<String,Integer> couponMap = new HashMap<String,Integer>();
    AggregationOutput productOutput = productCollection.aggregate(group);
    for (DBObject result : productOutput.results()) {
        BasicDBObject id = (BasicDBObject) result.get("_id");
        productMap.put(id.get("productCategory").toString(), Integer.parseInt(result.get("count").toString()));
    }
    DBCollection couponCollection = configDB.getCollection("coupons");
    DBObject group1 = new BasicDBObject("$group", new BasicDBObject("_id",new BasicDBObject("category", "$category")).append("count", new BasicDBObject("$sum", 1)) );
    AggregationOutput couponOutput = couponCollection.aggregate(group);
    for (DBObject result : couponOutput.results()) {
        BasicDBObject id = (BasicDBObject) result.get("_id");
        couponMap.put(id.get("category").toString(), Integer.parseInt(result.get("count").toString()));
    }
    BasicDBObject aggregate;
    for(String key: productMap.keySet()){
        aggregate = new BasicDBObject();
        aggregate.put("Category", key);
        aggregate.put("productCount", productMap.get(key));
        if(couponMap.containsKey(key))
            aggregate.put("couponCount", couponMap.get(key));
        else
            aggregate.put("couponCount", 0);
        response.add(aggregate);
    }
    for(String key:couponMap.keySet()){
        if(productMap.containsKey(key))
            continue;
        else{aggregate = new BasicDBObject();
            aggregate.put("Category", key);
            aggregate.put("productCount", 0);
            aggregate.put("couponCount", couponMap.get(key));
            response.add(aggregate);
        }
    }
    for(Object obj:response)
        return response;
```

Get Product – Code to display products on the Grid.

```
    @RequestMapping(value = "/getProducts", method = RequestMethod.GET)
    public @ResponseBody String getProducts(Locale locale, Model model) {
        logger.info("Fetching products from database:");

        MongoClient mongoClient = MongoConnection.getNewMongoClient();

    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("products");

    DBCursor profileDoc = collec.find(new BasicDBObject());
    Gson gson = new Gson();
    ArrayList<DBObject> products = new ArrayList<DBObject>();
    while(profileDoc.hasNext())
    {
        products.add(profileDoc.next());
    }

    return gson.toJson(products);
}
```

post product – Code to add products. Input from the user is added to the database and then displayed on the User Interface.

```
@RequestMapping(value = "{uid}/product", method = RequestMethod.POST, consumes = "application/json")
public @ResponseBody int add_prod(
        @RequestBody Product prod, Model model,
        HttpServletResponse response,@PathVariable ("uid") String uid)
        throws UnknownHostException {
    String name = prod.getProductName();
    String description = prod.getProductDescription();
    String category = prod.getProductCategory();
    String thumbnailImage = prod.getThumbnailImage();
    String url = prod.getProductUrl();
    String price = prod.getPrice();
    MongoClient mongoClient = MongoConnection.getNewMongoClient();
    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("products");
    //add to database
    BasicDBObject doc = new BasicDBObject();
    doc.put("productName", name);
    doc.put("productDesctiption", description);
    doc.put("productCategory", category);
    doc.put("thumbnailImage", thumbnailImage);
    doc.put("productUrl", url);
    doc.put("price", price);
    doc.put("uploadedBy", uid);
    DBCollection collec1 = configDB.getCollection("counters");
    BasicDBObject query = new BasicDBObject("_id", "productId");
    BasicDBObject sort = new BasicDBObject();
    BasicDBObject update = new BasicDBObject("$inc", new BasicDBObject("sequenceValue", 1));
    BasicDBObject fields = new BasicDBObject();
    System.out.println(collec1.findOne());
    DBObject productid = collec1.findAndModify(query, fields,sort, false, update, true, false);
    String id =  productid.get("sequenceValue").toString();
    int pid = (int) Float.valueOf(id).floatValue();
    System.out.println(pid);
    doc.put("_id", pid);
    collec.insert(doc);
    return pid;

}
```

update product – Code to update any product. Modal of Edit product will have all the fields pre-filled which is fetched from the database. If the user edits any field, changes are updated to the database as well as UI.

```
@RequestMapping(value = "{uid}/product/{id}", method = RequestMethod.PUT, consumes = "application/json")
public void edit_product(
        @RequestBody Product product, Model model,
        HttpServletResponse response, @PathVariable String uid, @PathVariable int id)
        throws UnknownHostException {

    BasicDBObject find = new BasicDBObject();
    BasicDBObject update = new BasicDBObject();
    find.put("_id", id);
    String name = product.getProductName();
    String description = product.getProductDescription();
    String category = product.getProductCategory();
    String thumbnailImage = product.getThumbnailImage();
    String url = product.getProductUrl();
    String price = product.getPrice();
    MongoClient mongoClient = MongoConnection.getNewMongoClient();
    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("products");
    //add to database
    BasicDBObject doc = new BasicDBObject();
    doc.put("productName", name);
    doc.put("productDesctiption", description);
    doc.put("productCategory", category);
    doc.put("thumbnailImage", thumbnailImage);
    doc.put("productUrl", url);
    doc.put("price", price);
    doc.put("uploadedBy", uid);
    update.put("$set",doc);
    System.out.println(doc);
    collec.update(find, update);
}
```

Update product Java Script

```
  $scope.updateProduct = function (product){
     // product.category='test';
     // product.picture='test';
      id = product._id;
      delete product["_id"];
      var formData = angular.toJson(product);

      console.log(formData);
      $http.put('/'+$scope.username+'/product/'+id, formData,
              {headers: {'Content-Type': 'application/json'}
      }).success(function(response) {
            console.log(response);
      });

    $modalInstance.close();
  }

  $scope.cancel = function () {
    $modalInstance.dismiss('cancel');
  };
});
```

delete product- Code to delete product. It deletes the product from database first and then the deletion is reflected on the list of products

```
@RequestMapping(value = "/product/{id}", method = RequestMethod.DELETE, consumes = "application/json")
public void delete_product(
        HttpServletRequest request, Model model,
        HttpServletResponse response, @PathVariable int id)
        throws UnknownHostException{
    MongoClient mongoClient = MongoConnection.getNewMongoClient();
    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("products");
    BasicDBObject doc = new BasicDBObject();
    System.out.println(doc);
    doc.put("_id", id);
    collec.remove(doc);
    System.out.println(doc);
}
```

product pojo - Getter Setter methods for all the fields of products.

```
public class Product {

    private String productName;
    private String productDescription;
    private String productUrl;
    private String productCategory;
    private String thumbnailImage;
    private String price;
    private String uploadedBy;

    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    public String getProductDescription() {
        return productDescription;
    }
    public void setProductDescription(String productDescription) {
        this.productDescription = productDescription;
    }
    public String getProductUrl() {
        return productUrl;
    }
    public void setProductUrl(String productUrl) {
        this.productUrl = productUrl;
    }
    public String getProductCategory() {
        return productCategory;
    }
    public void setProductCategory(String productCategory) {
        this.productCategory = productCategory;
    }
    public String getThumbnailImage() {
        return thumbnailImage;
    }
```

save upload product js – Save and upload javascript code for the product

```
$scope.saveProduct = function (product) {

    product.uploadedBy=$scope.username;
    product.thumbnailImage='';

    var formData = angular.toJson(product);

    $http.post('/'+$scope.username+'/product', formData,
            {headers: {'Content-Type': 'application/json'}
    }).success(function(response) {
          console.log(response);
          $scope.upload($scope.pictures[0],response);
      });

  $modalInstance.close();
};

$scope.upload = function (file,productId) {
    console.log(file);
    $upload.upload({
        url: 'product/'+productId,
        file: file
    }).progress(function (evt) {
        var progressPercentage = parseInt(100.0 * evt.loaded / evt.total);
        console.log('progress: ' + progressPercentage + '% ' + evt.config.file.name);
    }).success(function (data, status, headers, config) {
        console.log('file ' + config.file.name + 'uploaded.');
    });
};
```

update prod js – Javascript code to update the product

```
  $scope.updateProduct = function (product){
     // product.category='test';
     // product.picture='test';
      id = product._id;
      delete product["_id"];
      var formData = angular.toJson(product);

      console.log(formData);
      $http.put('/'+$scope.username+'/product/'+id, formData,
              {headers: {'Content-Type': 'application/json'}
      }).success(function(response) {
            console.log(response);
      });

    $modalInstance.close();
  }

  $scope.cancel = function () {
    $modalInstance.dismiss('cancel');
  };
});
```

product image – Code to upload the image, save it and then display thumbnail image on the UI

```
@RequestMapping(value = "{uid}/product/{productId}", method = RequestMethod.POST)
public String fileUpload(HttpServletRequest request, @RequestParam("file") MultipartFile files, @PathVariable("uid") String uid,@P
        String filePath ="/Users/hardik/Documents/Folders/PriceKing/src/main/webapp/resources/images/"+productId;
        File file = new File(filePath);
        System.out.println(filePath);
        if (!file.exists()) {
            if (file.mkdir()){
                saveFile(files.getInputStream(),filePath+"/"+files.getOriginalFilename());
                String output = "File saved to server location : " + filePath;
                System.out.println(output);
            }else{
                saveFile(files.getInputStream(),filePath+"/"+files.getOriginalFilename());
                String output = "File saved to server location : " + filePath;
                System.out.println(output);
            }
        }
        MongoClient mongoClient = MongoConnection.getNewMongoClient();

        DB configDB = mongoClient.getDB("priceking");
        DBCollection collec = configDB.getCollection("products");
        BasicDBObject query = new BasicDBObject("_id",productId);
        BasicDBObject update = new BasicDBObject("thumbnailImage","resources/images/"+productId+"/"+files.getOriginalFilename());
        collec.update(query, new BasicDBObject("$set",update));
    Gson gson = new Gson();
    return gson.toJson(filePath);
}
private void saveFile(InputStream uploadedInputStream,
        String serverLocation) {
    try {
        OutputStream outpuStream = new FileOutputStream(new File(serverLocation));
        int read = 0;
        byte[] bytes = new byte[1024];
        outpuStream = new FileOutputStream(new File(serverLocation));
        while ((read = uploadedInputStream.read(bytes)) != -1) {
            outpuStream.write(bytes, 0, read);
        }
        outpuStream.flush();
        outpuStream.close();
    } catch (IOException e) {
```

product modal – Code to use 1 modal for add as well as edit purpose.

```
if(prodIndex == -1) {
    $scope.showSaveProduct = true;
    $scope.showEditProduct = false;
}else {
    $scope.showEditProduct = true;
    $scope.showSaveProduct = false;
    $scope.product = $scope.productsList[prodIndex];

}
```

Modaljsp – JSP page which displays the modal along with the fields which a user needs to fill in.

```
    <div ng-controller="ModalDemoCtrl">
<script type="text/ng-template" id="addCoupon.html">
    <div class="modal-header">
        <h3 class="modal-title">Edit Coupons</h3>
    </div>
    <div class="modal-body">
       <table class="table">
            <tr>
                <td style="width:260px"><input type="text" ng-model="coupon.name" class="form-control" placeholder="Name"></td>

            </tr>
            <tr>
                <td colspan="2"><textarea placeholder="Description" rows="3" ng-model="coupon.description" class="form-control"></textarea></,
            </tr>
            <tr>
                <td><input type="text" ng-model="coupon.originalPrice" class="form-control" placeholder="Original Price"></td>
                <td><input type="text" ng-model="coupon.discountedPrice" class="form-control" placeholder="Discounted Price"></td>
            </tr>
            <tr>
                <td><input type="text" ng-model="coupon.couponCode" class="form-control" placeholder="Coupon Code"></td>
                <td>
                    <div class="row">
                     <div class="col-md-12">
                         <p class="input-group">
                          <input type="text" class="form-control" datepicker-popup="{{format}}" ng-model="coupon.expiryDate" is-open="opened'
                            <span class="input-group-btn">
                              <button type="button" class="btn btn-default" ng-click="open($event)"><i class="glyphicon glyphicon-calendar">
                              </span>
                         </p>
                    </div>
                    </div>
                </td>
            </tr>
            <tr>
            </tr>
        </table>
    </div>
    <div class="modal-footer">
        <button class="btn btn-primary" ng-click="saveCoupon(coupon)">Save</button>
```

coupon get – Code to get the coupons displayed on the UI

```
    @RequestMapping(value = "/getCoupons", method = RequestMethod.GET)
    public @ResponseBody String getCoupons(Locale locale, Model model) {
        logger.info("Fetching coupons from database:");

        MongoClient mongoClient = MongoConnection.getNewMongoClient();

    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("coupons");
    BasicDBObject fields = new BasicDBObject();
    fields.put("_id", 0);
    fields.put("name", 1);
    fields.put("description", 1);
    DBCursor profileDoc = collec.find(new BasicDBObject(),fields);
    Gson gson = new Gson();
    ArrayList<DBObject> coupons = new ArrayList<DBObject>();
    while(profileDoc.hasNext())
    {
        coupons.add(profileDoc.next());
    }

    return gson.toJson(coupons);
}
```

coupon post- Code to add new coupons

```
@RequestMapping(value = "/coupon", method = RequestMethod.POST, consumes = "application/json")
public void add_coup(@RequestBody Coupons coup, Model model,
        HttpServletResponse response)
        throws UnknownHostException {
    String name = coup.getName();
    String description = coup.getDescription();
    MongoClient mongoClient = MongoConnection.getNewMongoClient();
    DB configDB = mongoClient.getDB("priceking");
    DBCollection collec = configDB.getCollection("coupons");
    //add to database
    BasicDBObject doc = new BasicDBObject();
    doc.put("name", name);
    doc.put("description", description);
    doc.put("originalPrice", coup.getOriginalPrice());
    doc.put("discountedPrice", coup.getDiscountedPrice());
    doc.put("couponCode", coup.getCouponCode());
    doc.put("expiryDate", coup.getExpiryDate());
    DBCollection collec1 = configDB.getCollection("counters");
    BasicDBObject query = new BasicDBObject("_id", "couponId");
    BasicDBObject sort = new BasicDBObject();
    BasicDBObject update = new BasicDBObject("$inc", new BasicDBObject(
    "sequenceValue", 1));
    BasicDBObject fields = new BasicDBObject();
    System.out.println(doc);;
    DBObject productid = collec1.findAndModify(query, fields,
    sort, false, update, true, false);
    String id =  productid.get("sequenceValue").toString();
    int pid = (int) Float.valueOf(id).floatValue();
    System.out.println(pid);
    doc.put("_id", pid);
    collec.insert(doc);
}
```

coupon pojo – Getter and Setter methods for the coupon fields

```java
public class Coupons {

    private String name;
    private String description;
    private double originalPrice;
    private double discountedPrice;
    private String couponCode;
    private Date expiryDate;

    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getOriginalPrice() {
        return originalPrice;
    }
    public void setOriginalPrice(double originalPrice) {
        this.originalPrice = originalPrice;
    }
    public double getDiscountedPrice() {
        return discountedPrice;
    }
    public void setDiscountedPrice(double discountedPrice) {
        this.discountedPrice = discountedPrice;
    }
    public String getCouponCode() {
        return couponCode;
```

coupon controller js – Javascript coupon controller which handles the events like modal, updation according to user input, etc.

```
app.controller('couponController', function ($scope, $modalInstance, couponsList,$http, couponEdit) {
        |
    $scope.open = function($event) {
        $event.preventDefault();
        $event.stopPropagation();

        $scope.opened = true;
      };
      $scope.format = 'dd-MMMM-yyyy';
      $scope.couponsList = couponsList;
      $scope.ok = function () {
        $modalInstance.close($scope.selected.item);
      };
      $scope.saveCoupon = function (coupon) {

          var formData = angular.toJson(coupon);

          console.log(formData);
          $http.post('/coupon', formData,
                  {headers: {'Content-Type': 'application/json'}
          }).success(function(response) {
                console.log(response);
             });

      };

      $scope.cancel = function () {
        $modalInstance.dismiss('cancel');
      };
    });

app.directive("jqdatepicker", function () {
      return {
        restrict: "A",
        require: "ngModel",
        link: function (scope, elem, attrs, ngModelCtrl) {
          var updateModel = function (dateText) {
            scope.$apply(function () {
              ngModelCtrl.$setViewValue(dateText);
            });
          };
          var options = {
            dateFormat: "dd/mm/yy".
```

Cloud Deployment Prerequisites:

- Maven
- Java
- Tomcat 7
- MongoDB

1) Install Maven
   sudo apt-get install maven2
   check maven is installed
   mvn –verison

2) Install Java

sudo apt-get update
java -version
sudo apt-get install default-jre
sudo apt-get install default-jdk
Check Java is installed properly
Java –version

3) Install Tomcat7

sudo apt-get update
sudo apt-get install tomcat7
http://your_ip_address:8080

4) Install MongoDB

- sudo apt-key adv --keyserverhkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
- echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
- sudo apt-get update
- sudo apt-get install -y mongodb-org
- sudo apt-get install -y mongodb-org=3.0.2 mongodb-org-server=3.0.2 mongodb-org-shell=3.0.2 mongodb-org-mongos=3.0.2 mongodb-org-tools=3.0.2
- sudo service mongod start

Check MongoDB is installed or not

mongo –version

5) War file deployment steps

- Copy war file on cloud instance
- sudo scp -i ~/Downloads/stormkeys.pem cmpe295B-1.0.0-BUILD-SNAPSHOT.war ubuntu@54.187.35.18:~/
- deploy war file in webapps directory
- sudo mv cmpe295B-1.0.0-BUILD-SNAPSHOT.war /var/lib/tomcat7/webapps/priceking.war

- restart tomcat
- sudo service tomcat7 restart

```
ubuntu@ip-172-31-15-71:~$ java -version
java version "1.7.0_79"
OpenJDK Runtime Environment (IcedTea 2.5.5) (7u79-2.5.5-0ubuntu0.14.04.2)
OpenJDK 64-Bit Server VM (build 24.79-b02, mixed mode)
```

```
ubuntu@ip-172-31-15-71:~$ mvn -version
Apache Maven 2.2.1 (rdebian-14)
Java version: 1.7.0_79
Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux" version: "3.13.0-48-generic" arch: "amd64" Family: "unix"
```

```
ubuntu@ip-172-31-15-71:~$ mongo -version
MongoDB shell version: 3.0.2
```

```
ubuntu@ip-172-31-15-71:~$ sudo service tomcat7 status
 * Tomcat servlet engine is running with pid 1125
```

```
ubuntu@ip-172-31-15-71:~$ sudo service mongod status
mongod start/running, process 980
```

Mobile:

The mobile section consists of following system component API's for Android application:

SQLite Database:

PriceKing uses SQLite database to cache data to display recently viewed products and products that are added to wish list based on user credentials. It also uses SQLite database to cache the list of products and its images in order to perform different operations on it. SQLite is a lightweight database specifically for mobile devices. SQLite database stores data in such a way

that it can be accessed by only the application using it and not by any other application providing application level security by hiding data by other applications.

Shared Preferences:

PriceKing uses shared preferences to store the username of an authenticated user once signed in. Shared Preferences are the part of Android framework that is used to store primitive data types such as Integers, Strings, double etc.

Android Google Analytics:

Google analytics is basically a service offered by Google that is used for generating the detailed statistics about your applications or website. We are specifically using this in our Android application in order to get various user statistics about our application usage. The Google Analytics in our application enables following statistics about our deal finder Android app called PriceKing.

- App Installations around the globe.
- Active users and demographics (their geographical location)
- Screens and user engagements
- Crashes and exceptions

Google Play:

Google Play is the point of distribution of Android Apps. It allows all the users to view and download your Android applications previously known as Android Market. This is that one place where you can download your app from anywhere with the Internet connection enabled in your Android device. The PriceKing is live on Google Play and users can download our app using the following link:

Google Play Link:

https://play.google.com/store/apps/details?id=com.priceking

List Views:

List view is an Android component that displays the content mostly in the form of single column tabular format. We are using list view in our Android app to display all the Product results searched by the user. List View is used in displaying product result, recently searched products and products in the wish list. User can scroll down or up depending on the number of items in the list view in order to view all the product results. A list adapter holds the list view. We have used a list view adapter that extends base adapter.

Gesture Detection:

In the Recently Viewed Product detail and Wish List detail screen, we use View Pager component for viewing other product details by swiping left or right, which requires gesture detection mechanism in place. As a result, we make our activity extend gesture listener interface. This interface has onFling() method that we override. This method helps us identifying gesture whether it to the right of the screen or to the left of the screen. We accordingly put our logic and implement this module.

Web View:

Web View is an Android component that allows user to open any URL or HTML content to open inside it rather than moving outside the app and opening the content in a mobile browser. We make use web view client to open further links in web view itself. Although in this app we ask user whether to open the product purchase link in web view or browser. We provide both facilities to open the app in a web view as well as in the browser.

Action Bar:

Action bar is used for screen navigation in Android apps and it is available from Android 3.0 and above. It facilitates easy navigation from one activity to other or rather one screen to other from common users perspective.

Animations:

Android provides various animation techniques to improve user experience and keep user engaged to your app. We have used some of the animation techniques in our app. The animation in Android is provided in the xml file that is kept in the anim folder, which resides under the resource folder in the project structure. Some of the Animation techniques we used are as follows:

Translation*:*

The translation animation uses translation motion specified in the xml form. We have used this *translation* effect in the list view of product list while scrolling to provide better effect and user experience on UI.

Progress Dialog:

We are following a communication model where the UI hits the web service call. This web service call is executed on the background thread and the response is received and displayed on the main thread. Till then the main thread (UI thread) displays the progress dialog-asking user to wait until the response is received from the server.

Menu Drawer Library:

We have used an external Menu Drawer library for displaying menu on dragging from left hand side of the app. It provides convenience to the user while navigating from one feature to other and beautifies the user interface with its look. The following is the link for the library that we referred:

Menu Drawer Library Link:

https://github.com/SimonVT/android-menudrawer

OCR Library:

We have used an OCR (Optical Character Recognition) library to perform character recognition of the product images captured through camera of the Android device. The API uses Surface View on which it captures the image and perform character recognition along with displaying the part on the surface view that was recognize. The following is the link of the library that we used for performing OCR.

OCR Library Link:

https://github.com/rmtheis/android-ocr

Gallery & View Pager:

We have used Gallery and View Pager to enhance the user experience in the application. The app uses Gallery to display advertisements of different products of the users interest at Home page. The app uses View Pager to display product details of the recently viewed and wishlist products. The app also uses Cover Flow, which is an enhanced controller extended from gallery that displays provided images along with their reflection just below them.

Setting up Single node Storm Cluster on a t2.medium AWS instance:

Storm Kafka topology requires a high-end configuration to run in the production environment. The existing binary requires some stringent configuration and dependencies, so we have decided to build our own single node storm cluster from source.

The cluster configuration follows following steps

1. Creating a t2.medium instance on AWS

2. Select 2 cpu 4 gb memory machine to handle storm and Kafka installations. I have experienced some issues with Kafka installation with micro instances so preferred medium instance.

Step 2: Choose an Instance Type

Currently selected: t2.medium (Variable ECUs, 2 vCPUs, 2.5 GHz, Intel Xeon Family, 4 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
☑	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate

3. Keep the Step 3: Configure Instance Details to default and click next.

4. Add 30 gb storage for the instance as Kafka and Storm logs are decently bigger in size and will consume the memory real quick.

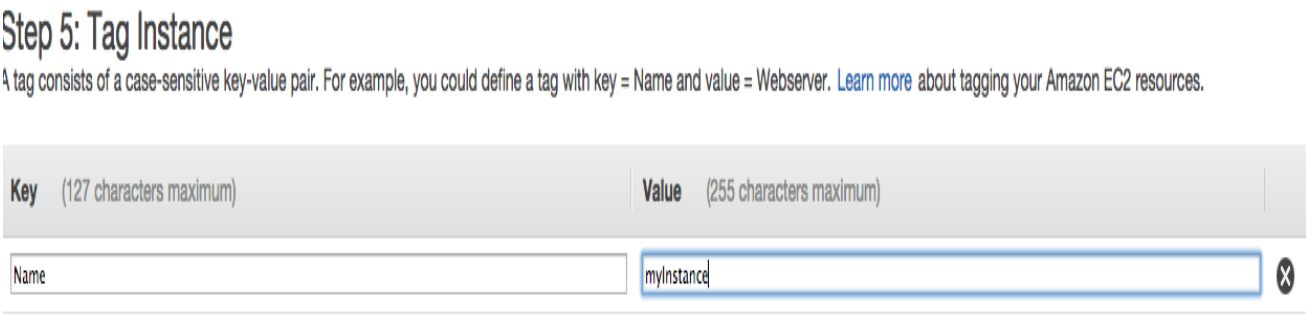
Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

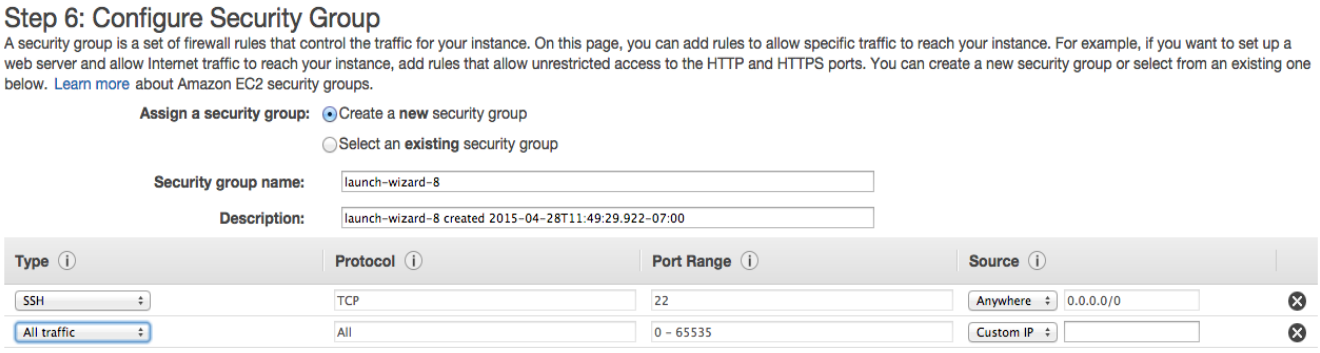
Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-bd9c25fb	30	General Purpose (SSD)	24 / 3000	☑	Not Encrypted

Add New Volume

5. Next tag the instance with any name



6. Next Setup security group, for installation purpose we will allow all traffic but after installation is done we can edit and restrict the security of the instance.



7. Launch the instance

Installing prerequisites for Storm

1. Update Ubuntu **$ sudo apt-get update**
2. Install JDK 7 **$sudo apt-get install openjdk-7-jdk**
3. Install git **$ sudo apt-get install git -y**
4. Install libtool **$ sudo apt-get install libtool -y**
5. Install Automake **$ sudo apt-get install automake -y**
6. Install uuid **$ sudo apt-get install uuid-dev**
7. Install g++ **$sudo apt-get install g++ -y**
8. Install gcc multilib **$ sudo apt-get install gcc-multilib –y**

Installing Zookeeper

"Storm uses Zookeeper for coordinating the cluster. Zookeeper is not used for message passing, so the load Storm places on Zookeeper is quite low. Single node Zookeeper clusters should be sufficient for most cases, but if you want failover or are deploying large Storm clusters you may want larger Zookeeper clusters."

1. Download the zookeeper from the following link you can use any latest zookeeper version. Create a directory in your home folder and put your configurations under it.

wget[**http://www.eng.lsu.edu/mirrors/apache/zookeeper/stable/zookeeper-3.4.6.tar.gz**](http://www.eng.lsu.edu/mirrors/apache/zookeeper/stable/zookeeper-3.4.6.tar.gz)

2. Extract it

root@ubuntu:/home/storm/zookeeper#

root@ubuntu:/home/storm# tar -xzf zookeeper-3.4.6.tar.gz

3. Setup zoo.cfg

root@ubuntu:/home/storm# cd zookeeper

root@ubuntu:/home/storm/zookeeper# cd con

conf/ contrib/

root@ubuntu:/home/storm/zookeeper# cd conf/

root@ubuntu:/home/storm/zookeeper/conf# nano zoo.cfg

put following lines in the file

tickTime=2000

dataDir=/home/storm/zookeeper-data

clientPort=2181

save and close the file

4. Create a zookeeper-data directory

root@ubuntu:/home/storm# mkdir zookeeper-data/

5. setup the zookeeper home in bashrc. I have added STORM_HOME AND PATH update already so please watch out for your path and update the path accordingly

$ vi ~/.bashrc

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64

export ZOOKEEPER_HOME=/home/storm/zookeeper

export STORM_HOME=/home/storm/storm

export PATH=$PATH:$JAVA_HOME/bin:$ZOOKEEPER_HOME/bin:$STORM_HOME/bin

6. zookeeper is set up now and try running it

cd /home/storm/zookeeper/bin

./zkServer.sh start

JMX enabled by default

Using config: /home/storm/zookeeper/bin/../conf/zoo.cfg

Starting zookeeper ... STARTED

Installing ZeroMQ

1. Storm internally uses ZeroMQ and for the source setup we need to explicitly set it up.
2. Download ZeroMQ **$ wget http://download.zeromq.org/zeromq-2.1.7.tar.gz**
3. Untar the binary **$tar –xzf zeromq-2.1.7.tar.gz**
4. Go to the ZeroMQ directory **$cd zeromq-2.1.7**
5. Configure zeromq **$ ./configure**
6. **$make**
7. **$ sudo make install**

Installing java binding for ZeroMQ

1. **clone the git repository in the same directory of installations**
2. **$ git clone https://github.com/nathanmarz/jzmq.git**
3. This will create a folder named jzmq
4. **Go to the directory cd /jzmq**
5. **$sed -i 's/classdist_noist.stamp/classnoist.stamp/g' src/Makefile.am**
6. **$ ./autogen.sh**
7. **$ ./configure**

8. *if you get an error message like "*** No rule to make target `classdist_noinst.stamp', needed by `org/zeromq/ZMQ.class'. Stop. make[1]: Leaving directory `/home/localadmin/jzmq/src'"* **then follow the steps a,b,c else follow 9**
   a. **cd jzmq/src**
   b. **sudo touch classdist_noinst.stamp**
   c. **sudo CLASSPATH=.:./.:$CLASSPATH javac -d . org/zeromq/ZMQ.java org/zeromq/ZMQException.java org/zeromq/ZMQQueue.java org/zeromq/ZMQForwarder.java org/zeromq/ZMQStreamer.java**
   d. **sudo nano configure**
   e. add the following line after any export line **export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64**

9. **$ ./configure**
10. **$ make**
11. **$ sudo make install**

Installing Storm

1. Get the storm binary from following link

$ wget http://mirror.tcpdiag.net/apache/incubator/storm/apache-storm-0.9.1-incubating/apache-storm-0.9.1-incubating.tar.gz

2. Untar the binary **$ tar -xvf** apache-storm-0.9.1-incubating.tar.gz

3. Add STORM_HOME to bashrc and update the path as mentioned above

4. Create a storm data folder **$mkdir data**

5. Update the storm.yaml with following configurations **$cd /home/storm/apache-storm-9.0.1/conf/**

6. add following lines to storm.yaml if you want to create a new storm.yaml file else change existing storm.yaml with following configurations.

storm.zookeeper.servers:

- "127.0.0.1"

storm.zookeeper.port: 2181

nimbus.host: "127.0.0.1"

nimbus.thrift.port: 6627

ui.port: 8772

storm.local.dir: "/home/storm/storm/data"

java.library.path: "/usr/lib/jvm/java-6-openjdk-amd64"

supervisor.slots.ports:

- 6700

- 6701

- 6702

- 6703

7. Go to STORM_HOME/bin **$cd /home/storm/storm/bin**

8. Start Nimbus **./storm nimbus**

9. open another terminal window go to the storm/bin directory and Start Supervisor **./storm supervisor**

10. Open another terminal window and go to the storm/bin directory and start storm UI **./storm ui**

open the browser and type in http://127.0.0.1:8772 and check if Storm UI correctly displayed.

4.6Software Design Problems, Solutions, and Patterns

The design is under discussion and the patterns to be used are yet to be finalized. There would be asynchronous communication between the independent systems considering the performance issues.

The hardware requirement for the system to run uninterruptedly is quite stringent, so performance tuning is required for every component in our system.

5. System Implementation

5.1 System Implementation summary

Our system consists of 4 subsystems namely mobile, platform, database and user dashboard interface.

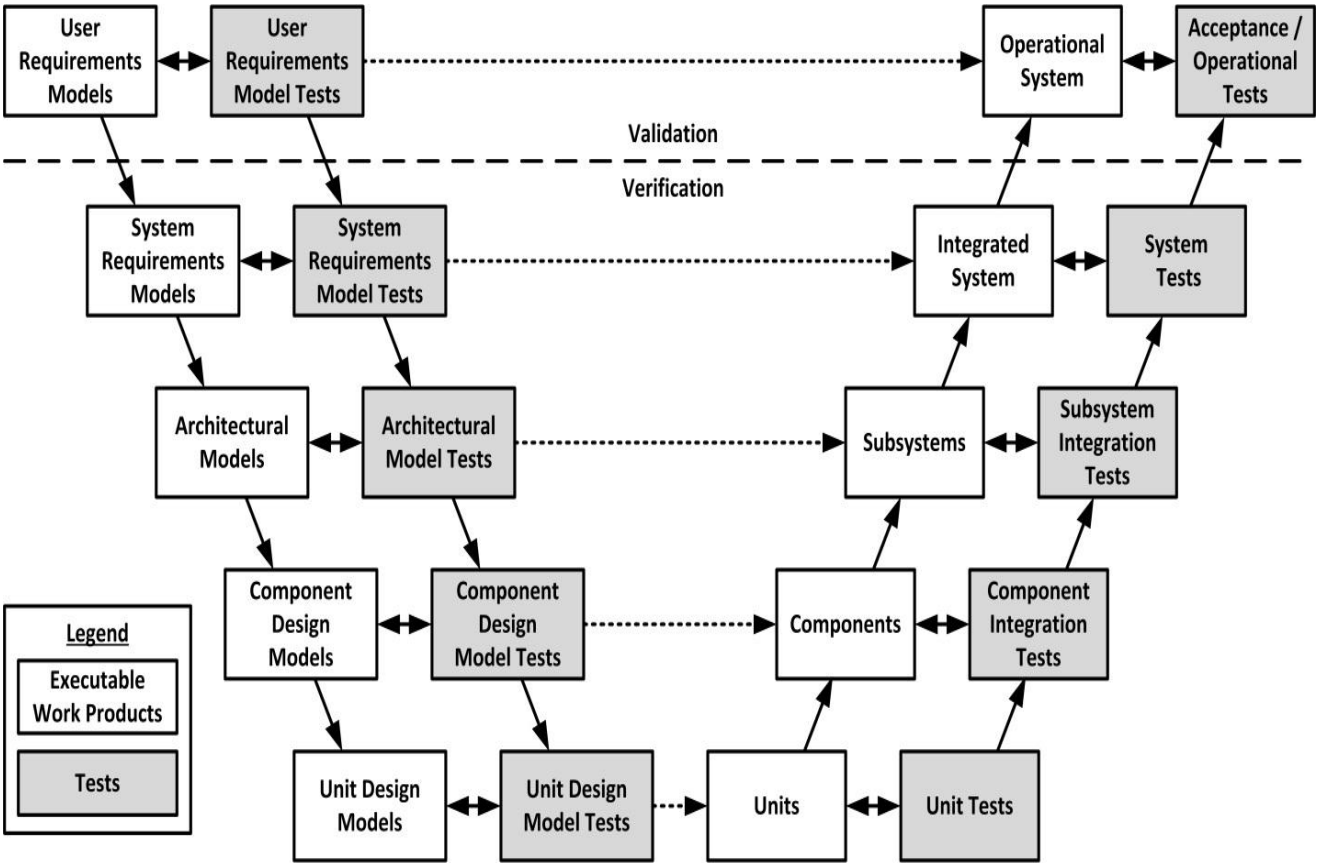


Figure 5.1: V Model

Each subsystem performs its tasks synchronously and relays the data into next subsystem where the incoming data is processed and filtered result set is sent back to the mobile request. Each subsystem is elaborated below

5.1.1 Mobile

Mobile Services included Android SDK specified in the eclipse with Android ADT. Android SDK can be downloaded from the following link:

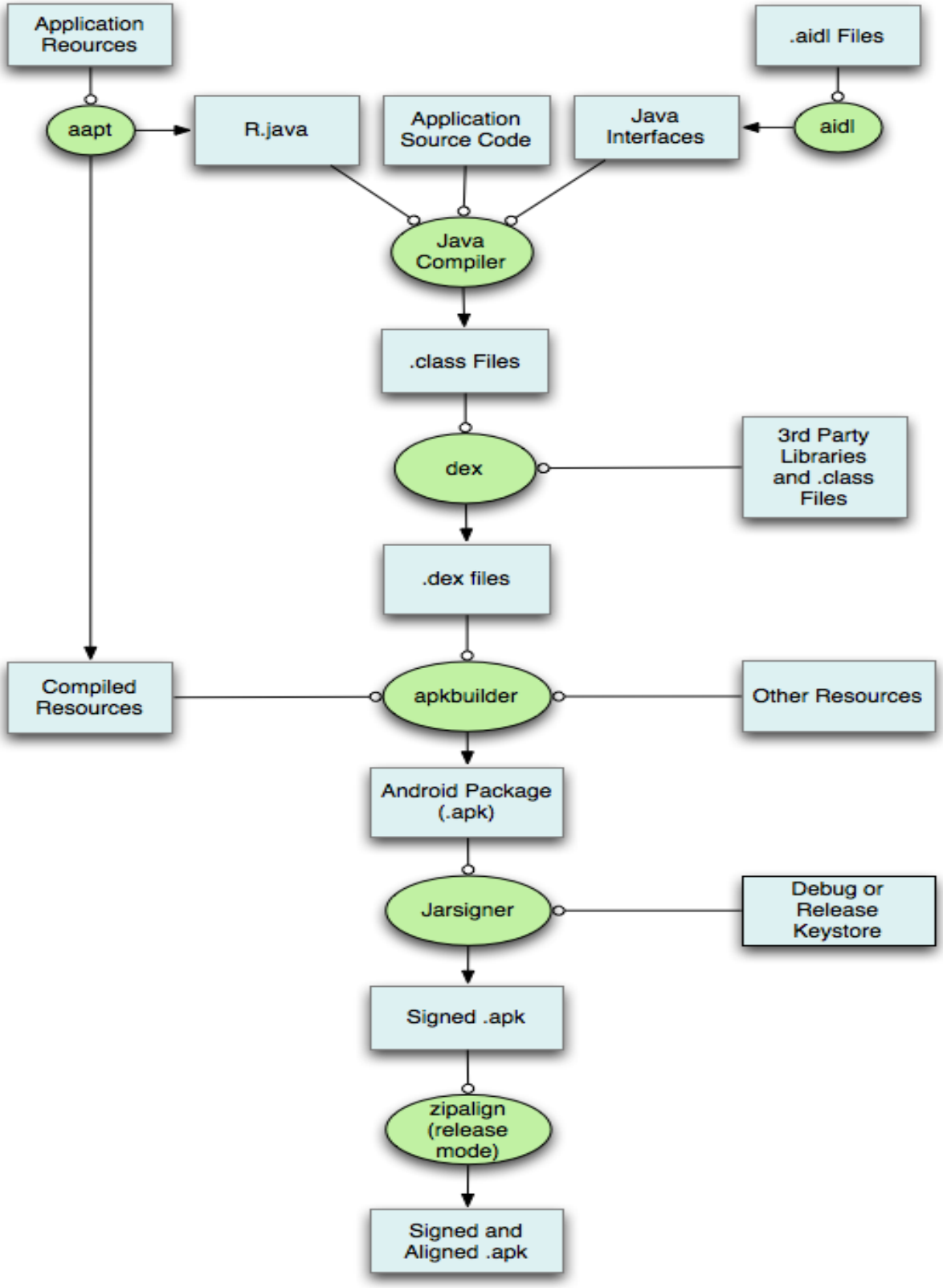
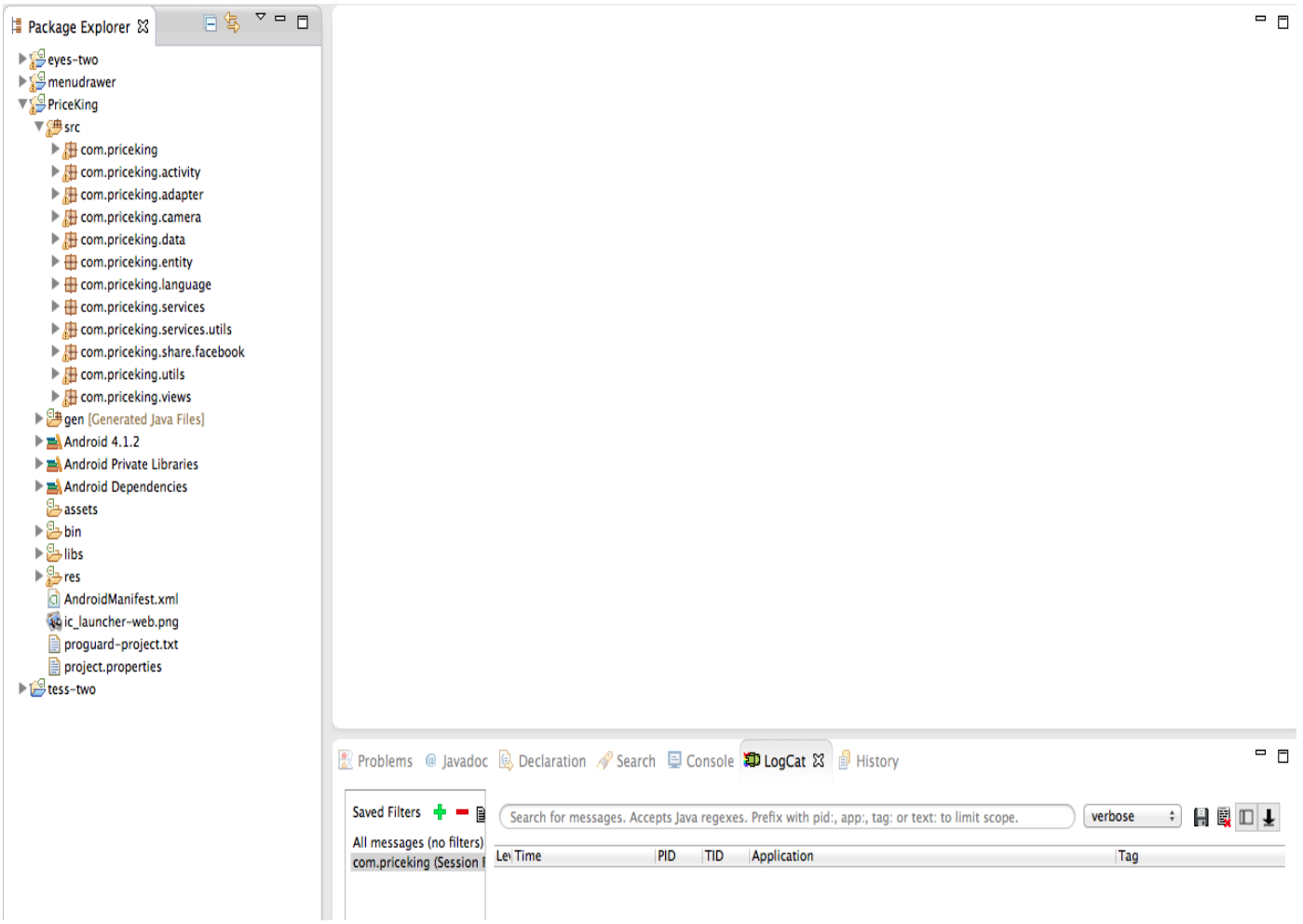


Figure 5.2 :Android Development Flow

Android SDK Link:

https://developer.android.com/sdk/index.html

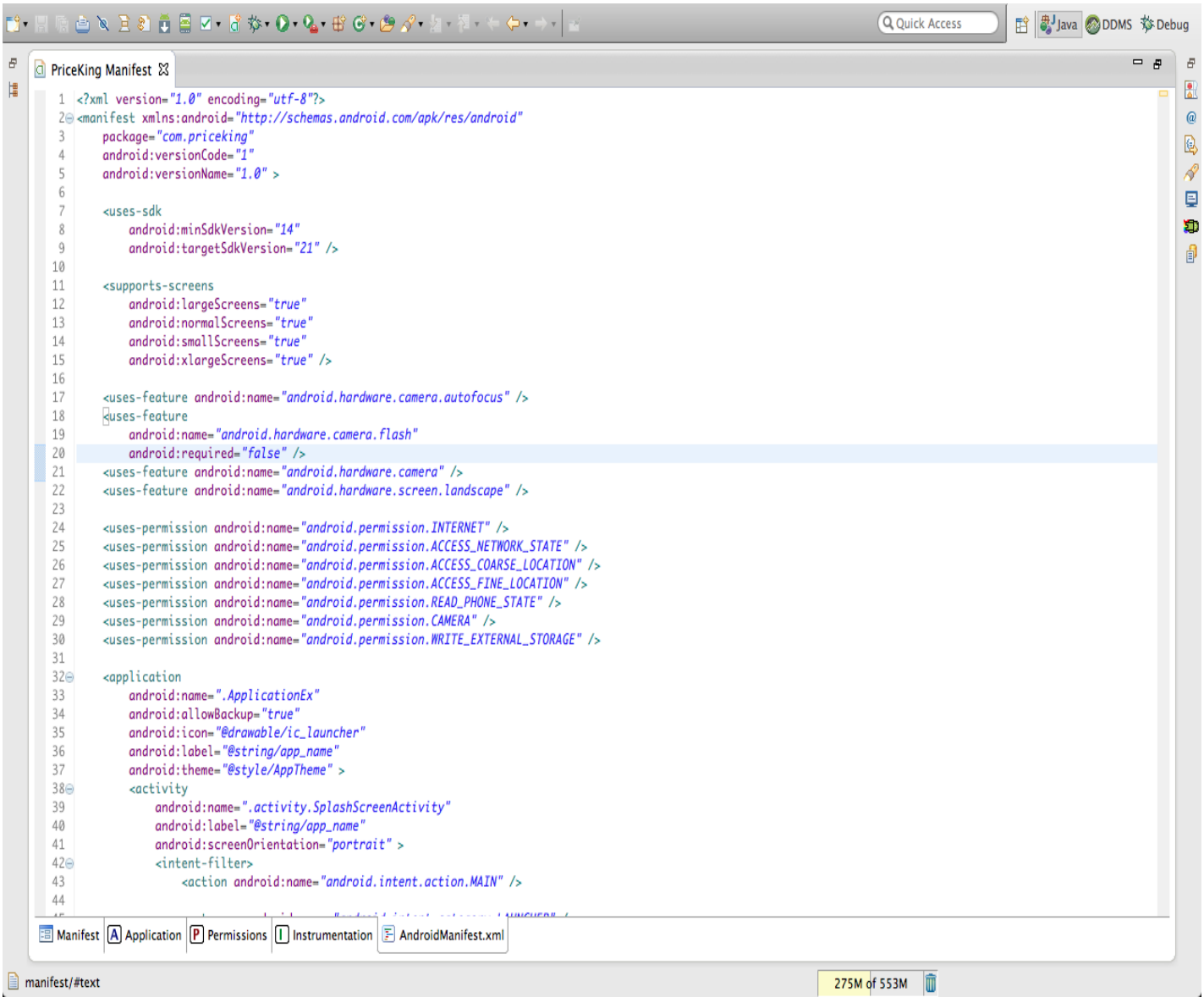
Once downloaded, we create an Android application called “PriceKing” in the eclipse. The project uses various UI libraries such as “Menu Drawer” for sliding menu options. It also uses an OCR library to get the characters from an image captured. The structure of the project of Android app looks like:



Android app-project structure

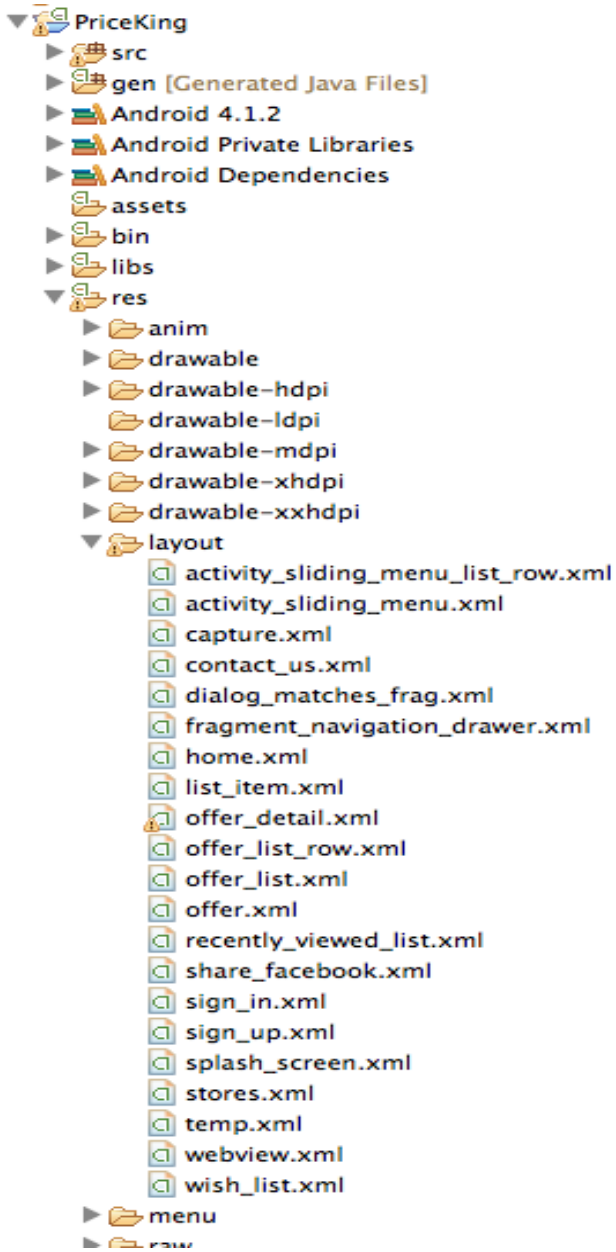
Android application has an AndroidManifest.xml file, that is used as a configuration file. If the app requires a new Activity, it has to be specified in this manifest file, else the app will crash throwing “Activity not found” exception. The manifest file also requires permission to access different features of mobile devices such as Camera, Internet, GPS etc.

AndroidManifest.xml:



Android manifest code snippet

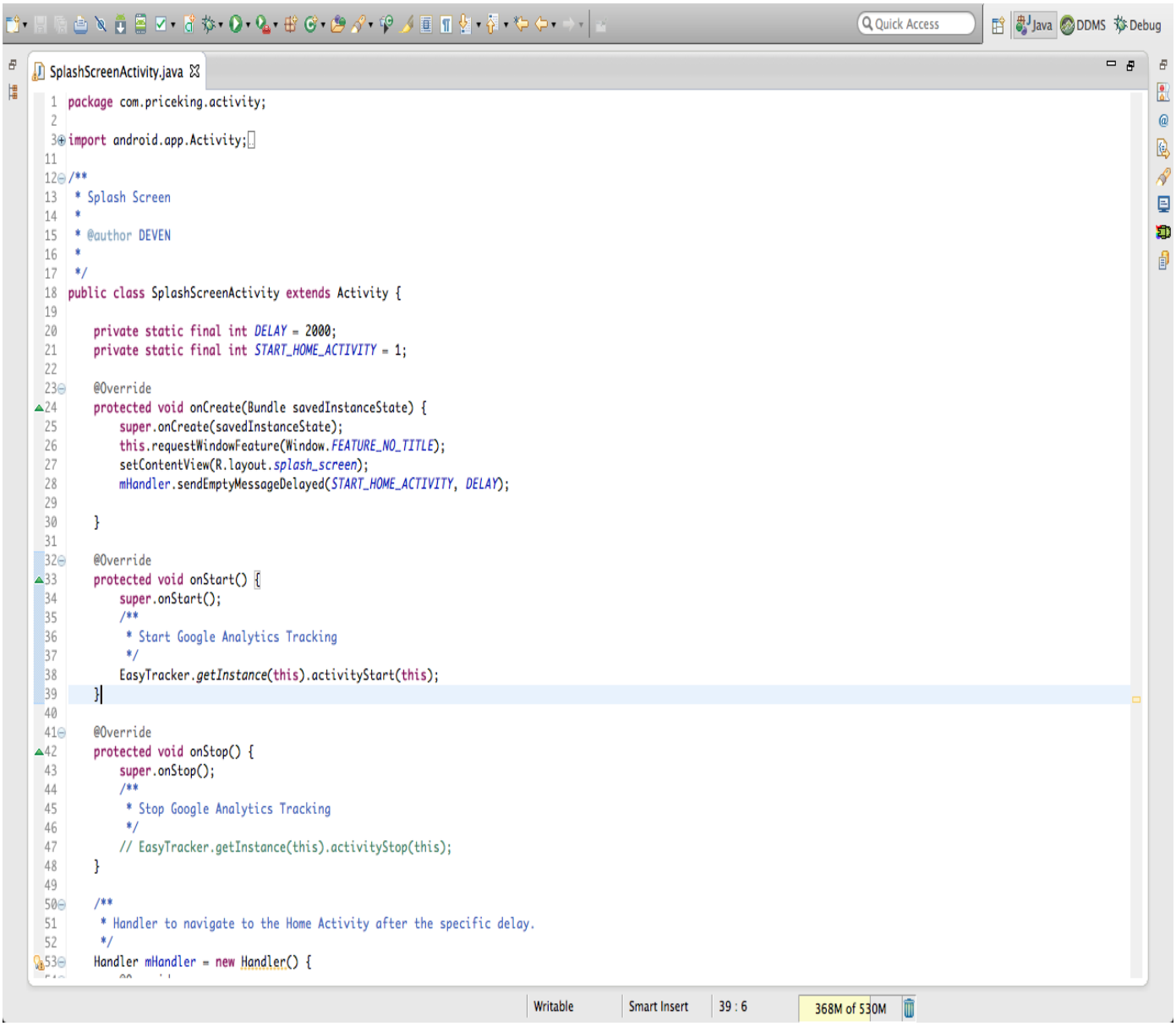
The application has different activities that are used for different functionalities. These Activities are associated with views that are mentioned under “res/layout” directory.



Android app class structure

The following is the sample of the activities and its user interface code snippet.

Splash Screen Activity:




```java
package com.priceking.activity;

import android.app.Activity;

/**
 * Splash Screen
 *
 * @author DEVEN
 *
 */
public class SplashScreenActivity extends Activity {

    private static final int DELAY = 2000;
    private static final int START_HOME_ACTIVITY = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.splash_screen);
        mHandler.sendEmptyMessageDelayed(START_HOME_ACTIVITY, DELAY);

    }

    @Override
    protected void onStart() {
        super.onStart();
        /**
         * Start Google Analytics Tracking
         */
        EasyTracker.getInstance(this).activityStart(this);
    }

    @Override
    protected void onStop() {
        super.onStop();
        /**
         * Stop Google Analytics Tracking
         */
        // EasyTracker.getInstance(this).activityStop(this);
    }

    /**
     * Handler to navigate to the Home Activity after the specific delay.
     */
    Handler mHandler = new Handler() {
```

Splash screen activity code snippet

Splash Screen Layout:

splash_screen.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/white"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/spash_image"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="fitXY"
        android:src="@drawable/splash_screen" />

</LinearLayout>
```

Splash screen layout code snippet

Code Snippet for Sliding Menu:

```
public MenuDrawer mMenuDrawer;

// Set Menu to the left of the screen

mMenuDrawer= MenuDrawer.attach(BaseActivity.this, Position.LEFT);

// Set touch Mode to drag

mMenuDrawer.setTouchMode(MenuDrawer.MENU_DRAG_WINDOW);

// Set the layout for sliding menu

mMenuDrawer.setMenuView(R.layout.activity_sliding_menu);

// Set the menu size after it is dragged on the screen
```

```
mMenuDrawer.setMenuSize(1000);


// Set Category List on Slider Menu listView = (ListView) findViewById(R.id.list); adapter = new
CategoryAdapter(this, PriceKingUtils.getCategoryList()); listView.setDrawSelectorOnTop(true);
listView.setAdapter(adapter); listView.setOnItemClickListener(onItemClickListner);

// Open the menu if it is closed

final int drawerState = mMenuDrawer.getDrawerState();

if (drawerState == MenuDrawer.STATE_CLOSED

|| drawerState == MenuDrawer.STATE_CLOSING) {

mMenuDrawer.openMenu();

}
```

Code Snippet for Gallery:

```
/*

* A structure describing general information about a display, such

* as its size, density, and font scaling.To access the

* DisplayMetrics members, initialize an object like this:

*/

DisplayMetrics metrics = new DisplayMetrics();

getWindowManager().getDefaultDisplay().getMetrics(metrics);

// Converts 14 dip into its equivalent px

Resources r = getResources();

float px = TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,

100, r.getDisplayMetrics());

// set gallery to left side

MarginLayoutParams mlp = (MarginLayoutParams) gallery.getLayoutParams();
```

```
mlp.setMargins((int) -(metrics.widthPixels / 2 + (px / 2)),
mlp.topMargin, mlp.rightMargin, mlp.bottomMargin);
AdvertisementGalleryAdapter adapter = new AdvertisementGalleryAdapter(
HomeActivity.this, advertisementList);
gallery.setAdapter(adapter);
// clicklistener for Gallery
gallery.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
// Save Product to Recently Viewed Items Table in Database
});
}
```

Code Snippet for Reminder of Deal:

```
// select date for adding reminder
Calendar cal = Calendar.getInstance(TimeZone.getDefault());
DatePickerDialog datePicker = new DatePickerDialog(
ProductDetailActivity.this, datePickerListener,
cal.get(Calendar.YEAR), cal.get(Calendar.MONTH),
cal.get(Calendar.DAY_OF_MONTH));
datePicker.setCancelable(false);
datePicker.setTitle("Select the date");
datePicker.show();
```

```
// add product to calendar
/**
* Add Event to Calendar
*/
public void addEventToCalendar(int year, int month, int day) {
Intent calIntent = new Intent(Intent.ACTION_INSERT);
calIntent.setType("vnd.android.cursor.item/event");
calIntent.putExtra(Events.TITLE, "Reminder: " + product.getName());
calIntent.putExtra(Events.DESCRIPTION, product.getShortDescription());
GregorianCalendar calDate = new GregorianCalendar(2012, 7, 15);
calIntent.putExtra(CalendarContract.EXTRA_EVENT_ALL_DAY, true);
calIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,
calDate.getTimeInMillis());
calIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
calDate.getTimeInMillis());
startActivity(calIntent);
}
```

Code Snippet for Sharing Deal:

```
// Example of Twitter Sharing
Intent tweetIntent = new Intent(Intent.ACTION_SEND);
tweetIntent.putExtra(Intent.EXTRA_TEXT, product.getProductURL());
tweetIntent.setType("text/plain");
```

```
packManager = getPackageManager();
resolvedInfoList = packManager.queryIntentActivities(tweetIntent,
PackageManager.MATCH_DEFAULT_ONLY);
resolved = false;
for (ResolveInfo resolveInfo : resolvedInfoList) {
if (resolveInfo.activityInfo.packageName
.startsWith("com.twitter.android")) {
tweetIntent.setClassName(
resolveInfo.activityInfo.packageName,
resolveInfo.activityInfo.name);
resolved = true;
break;
}
}
if (resolved) {
startActivity(tweetIntent);
} else {
   Toast.makeText(ProductDetailActivity.this,
   "Twitter app isn't found", Toast.LENGTH_LONG).show();
}
```

Code Snippet for Buy Now:

```
/**
* for opening the Product URL in browser.
*/
Intent myIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse(product.getProductURL()));
startActivity(myIntent);
```

Code Snippet for SQLite Database:

```
/**
* Database Version
*/
private static final int DATABASE_VERSION = 9;
/**
* Database Name
*/
private static final String DATABASE_NAME = "priceking.db";
```

```
/**
* Table Product
*/
public static final String TABLE_PRODUCT = "product";
```

```
public static final String TABLE_RECENTLY_VIEWED = "recently_viewed";
public static final String TABLE_WISH_LIST = "wish_list";
public static final String TABLE_ADVERTISEMENT = "advertisement";
public static final String TABLE_COUPON = "coupon";
private static final String KEY_PRODUCT_NAME = "NAME";
private static final String KEY_MSRP = "MSRP";
private static final String KEY_SALE_PRICE = "SALE_PRICE";
private static final String KEY_CATEGORY = "CATEGORY";
private static final String KEY_SHORT_DESCRIPTION = "SHORT_DESCRIPTION";
private static final String KEY_THUMBNAIL_IMAGE = "THUMBNAIL_IMAGE";
private static final String KEY_PRODUCT_URL = "PRODUCT_URL";
private static final String KEY_CUSTOMER_RATING = "CUSTOMER_RATING";
private static final String KEY_THUMBNAIL_BLOB = "THUMBNAIL_BLOB";
/**
* Creating Product Table
*/
String CREATE_PRODUCT_TABLE = "CREATE TABLE " + TABLE_PRODUCT + "("
+ KEY_ID + " INTEGER PRIMARY KEY," + KEY_PRODUCT_NAME
+ " TEXT," + KEY_MSRP + " INTEGER," + KEY_SALE_PRICE
+ " INTEGER," + KEY_CATEGORY + " TEXT," + KEY_SHORT_DESCRIPTION
+ " TEXT," + KEY_THUMBNAIL_IMAGE + " TEXT," + KEY_PRODUCT_URL
+ " TEXT," + KEY_CUSTOMER_RATING + " INTEGER,"
+ KEY_THUMBNAIL_BLOB + " BLOB" + ")";
db.execSQL(CREATE_PRODUCT_TABLE);
```

```
/**
* Opens the Database.
*/
public void openInternalDB() {
if ((m_db == null) || (m_db.isOpen() == false)) {
    m_db = this.getWritableDatabase();
}
}
/**
 * Closes the Database.
*/
public void closeDB() {
    if (m_db != null) {
m_db.close();
m_db = null;
}
}
/**
* Delete table entries of the table with given Table Name BASED ON QUERY
*
* @param TABLE_NAME
*/
public void deleteTableEntries(String TABLE_NAME, String productName) {
int rows = m_db.delete(TABLE_NAME, KEY_PRODUCT_NAME + "=?",
```

```
new String[] { productName });
System.out.println("****deleted rows******" + rows);
}
/**
* Adds Product to Database.
*/
public void addProduct(String TABLE_NAME, Product product) {
SQLiteDatabase db = this.getWritableDatabase();
ContentValues values = new ContentValues();
values.put(KEY_PRODUCT_NAME, product.getName());
values.put(KEY_MSRP, product.getMsrp());
values.put(KEY_SALE_PRICE, product.getSalePrice());
values.put(KEY_CATEGORY, product.getCategory());
values.put(KEY_SHORT_DESCRIPTION, product.getShortDescription());
values.put(KEY_THUMBNAIL_IMAGE, product.getThumbnailImage());
values.put(KEY_PRODUCT_URL, product.getProductURL());
values.put(KEY_CUSTOMER_RATING, product.getCustomerRating());
values.put(KEY_THUMBNAIL_BLOB, product.getThumbnailBlob());
long result = db.insert(TABLE_NAME, null, values);
System.out.println("Product Insertion Result" + result);
db.close();
}
```

```
/** * * @return product list from database. */ public List<Product> getproductList(String
TABLE_NAME) { List<Product> productList = new ArrayList<Product>(); String selectQuery =
"SELECT * FROM " + TABLE_NAME; SQLiteDatabase db = this.getWritableDatabase(); Cursor
cursor = db.rawQuery(selectQuery, null); if (cursor.moveToFirst()) { do { Product product = new
Product(); product.setName(cursor.getString(cursor
.getColumnIndex(DatabaseHandler.KEY_PRODUCT_NAME)));
product.setMsrp(cursor.getDouble(cursor .getColumnIndex(DatabaseHandler.KEY_MSRP)));
product.setSalePrice(cursor.getDouble(cursor
.getColumnIndex(DatabaseHandler.KEY_SALE_PRICE)));
product.setCategory(cursor.getString(cursor
.getColumnIndex(DatabaseHandler.KEY_CATEGORY)));
product.setShortDescription(cursor.getString(cursor
.getColumnIndex(DatabaseHandler.KEY_SHORT_DESCRIPTION)));
product.setThumbnailImage(cursor.getString(cursor
.getColumnIndex(DatabaseHandler.KEY_THUMBNAIL_IMAGE)));
product.setProductURL(cursor.getString(cursor
.getColumnIndex(DatabaseHandler.KEY_PRODUCT_URL)));
product.setCustomerRating(cursor.getDouble(cursor
.getColumnIndex(DatabaseHandler.KEY_CUSTOMER_RATING)));
product.setThumbnailBlob(cursor.getBlob(cursor
.getColumnIndex(DatabaseHandler.KEY_THUMBNAIL_BLOB))); productList.add(product); }
while (cursor.moveToNext()); } cursor.close(); db.close(); return productList; }
```

Coding Style:

Following is the recommended coding style.

- Always state the scope (private, public, etc) of classes, methods and class variables.
- With the exception of zero and one, never hard-code a numeric value; always declare a constant instead.
- Always use zero-based arrays and with indexed collection, use zero-based indices.
- All the string visible to end-user should be placed in res/strings.xml for ease of localization.

- Use StringBuilder for building long strings.
- Use TextUtils.isEmpty(String str) for checking empty strings as it verifies the null condition as well.
- Always use a default case in a switch statement that asserts.
- Prefer generic flavor before non-generic if both implementations are available.
- Use application logging and debugging.
- Remove logs from the code while releasing build for google play.
- Don't make any instance or class variable public without good reason.
- Avoid using an object to access a class (static) variable or method. Use a class name instead.
- For call backs from the network layer to UI layer use interfaces or handler (ANDROID specific).
- Colors used within the project must be in color.xml file of /res/values.
- Menu (commands) for each activity must be in menu.xml file of /res/values.
- Avoid using splash screen for applications as separate launcher activities; use it along with the first screen to be displayed as a separate view.
- Whenever you use layout_margins please check the effects of orientation change (Landscape -> Potrait and Potrait -> Landscape) for the same.
- Any control in the screen should have equal margin from all the sides.
- While creating databases make sure to use less no. of columns or merge few of them in to a single column. Consider using binary operators like & for the same. More on binary operators on the below linkhttp://www.mssqltips.com/tip.asp?tip=1218.
- In case using databases always try using managedQuery for firing a query to the DB as the entire query maintenance is handled by the Android system itself i.e. closing and deactivating the cursor when the Activity class is destroyed. The managedQuery is available only for the Activity class. In case simple query is used on a cursor object make sure the cursor object is deactivated, closed and set to null in order to avoid memory leaks.

- All the blocking calls must be done on a separate thread and not on the UI thread, so that it doesn't block the UI.
- Always use dip (Density independent pixels) in layout xml files for defining height and width of the component.
- Use sp for defining height of the text used.
- Use px to define width/height whenever exact size component required in pixels.
- Understand the difference between "GONE" and "INVISIBLE" for a view.
- Manipulating any UI operation from the network thread must be done in a UI thread only example updating a listview once the data is fetched from the network. The example for the same are as below:
- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable)
- Handler
- AsyncTask
- Always use layout files to create user interface unless and until there is specific requirement.
- Temporary files created must be cleared at interval or every app termination.
- Progress dialog should behave same in the entire app.
- Should handle connection related issues like, no internet connection, server connection time out.
- While making network connection, explicitly mention request type(POST ,GET)
- Encode all the data which is sent with the request with URL encoding.
- Direct network calls must be avoided from UI(Activity) classes. Responsibility for network related calls and the response handling must be delegated to the Manager (Controller) classes.
- Create generic data models for data received from the network.
- Parsing of the data should be done in entity.

Design Pattern:

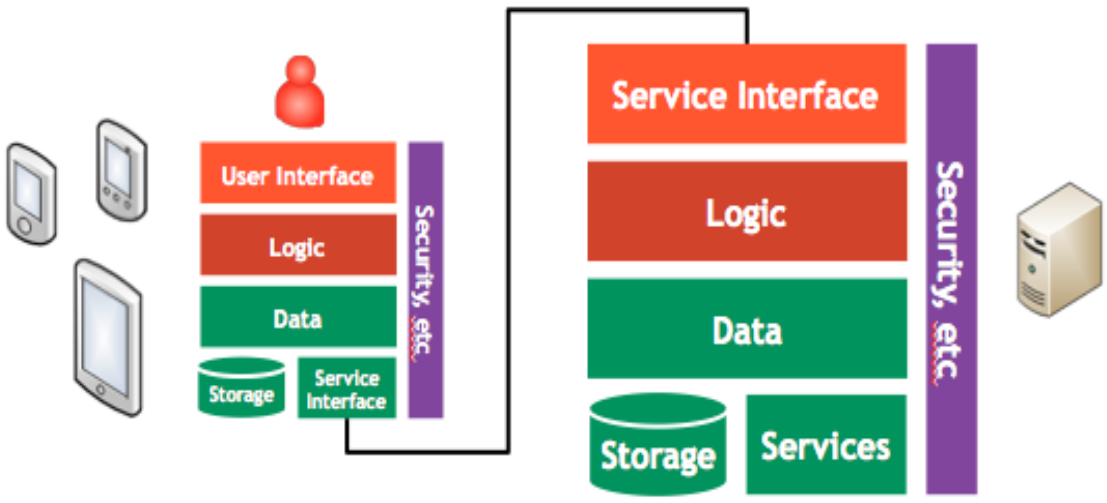
Builder Design Pattern for Storm implementation

The key feature of Builder pattern is that it involves a step-by-step process to build something, i.e., every produce will follow the same *process* even though each step is different.

Singleton Design Pattern

```
private static DBCollection sessionsCollection;

private static SessionDAO instance;

public SessionDAO(final DB projectDB) {
    sessionsCollection = projectDB.getCollection("session");
}

public static SessionDAO getInstance() throws UnknownHostException {
    if(instance == null){
        MongoClient mongoClient = new MongoClient(new ServerAddress(
                "localhost", 27017));
        final DB pricekingDB = mongoClient.getDB("priceking");
        instance = new SessionDAO(pricekingDB);
    }
    return instance;
}
```

```
private static UserDAO instance;
private final DBCollection usersCollection;

private Random random = new SecureRandom();

public UserDAO(final DB projectDB){
    usersCollection = projectDB.getCollection("profiles");
}

 public static UserDAO getInstance() throws UnknownHostException {
        if(instance == null){
            MongoClient mongoClient = new MongoClient(new ServerAddress(
                    "localhost", 27017));
            final DB pricekingDB = mongoClient.getDB("priceking");
            instance = new UserDAO(pricekingDB);
        }
        return instance;
    }
```



Architecture Tiers

The design patterns used in this app can be divided into following four sections:

1. Front End:

The role of the user interface components, views, is to provide the best user experience possible. This is a huge subject, and not within the scope of this document, but it's not only about something that looks nice, but also feels good, something Tapworthy. The views should know about each other, they should know about data in the form of entities, and they should know about logic components (domains) or might know about services (in simpler apps). But the views should never know about networks, URLs, protocols like JSON, or even key-value pairs derived directly from JSON. They shouldn't have to worry about how data is persistently stored or cached, or any lower-level logic.

2. Entities:

The entities are the representation of data in a form that makes sense to the app. It may have the exact same structure as the entities retrieved from the server (when a Mobile Service is in play one can come really close), but most often or may have a different structure that better suit the app. For example the app's entities may add attributes that are only valid for the app (e.g. a flag that something is a favorite, if favorites are only stored on the device). The entities also know how to parse the raw data (JSON) from the server to populate its attributes. That also includes any related entities (e.g. an order entity will probably use order row entities to parse their data). The entities will probably not implement much logic, with the exception of derived attributes and any entity-related rules, and should not know anything about either the user interface or what the services knows (network requests, etc).

3. Middle Tier:

Service components make network calls to REST based (web) services using on the server (preferably a Mobile Service). They handle all the logic that is involved in network request, like HTTP returns codes, authentication, exceptions, caching, etc. They also initiate the parsing of data (JSON) to entities. Just as the views should not know anything about what services know, the services should not know anything about the user interface or entity internals.

4. Data Store:

The idea behind the domain components is to implement the logic of a specific business domain. It could be an order, and then the domain would know about a number of services and entities, e.g. order details, order rows, customer, delivery address, etc. Domains will manage both the data and logic related to the cluster of entities and services that form the (business) domain. This also includes the caching and refresh of that data, using any kind of storage (database, file system, etc). Views use domains to get the data and implement the logic that they need, but domains doesn't know anything about the user interface.

When apps are very simple and don't implement a lot of logic, it's possible to create a singleton domain (often simply called "Data") that hold data and minor logic common to the whole app.

Google Analytics SDK for Android V3 (Legacy):

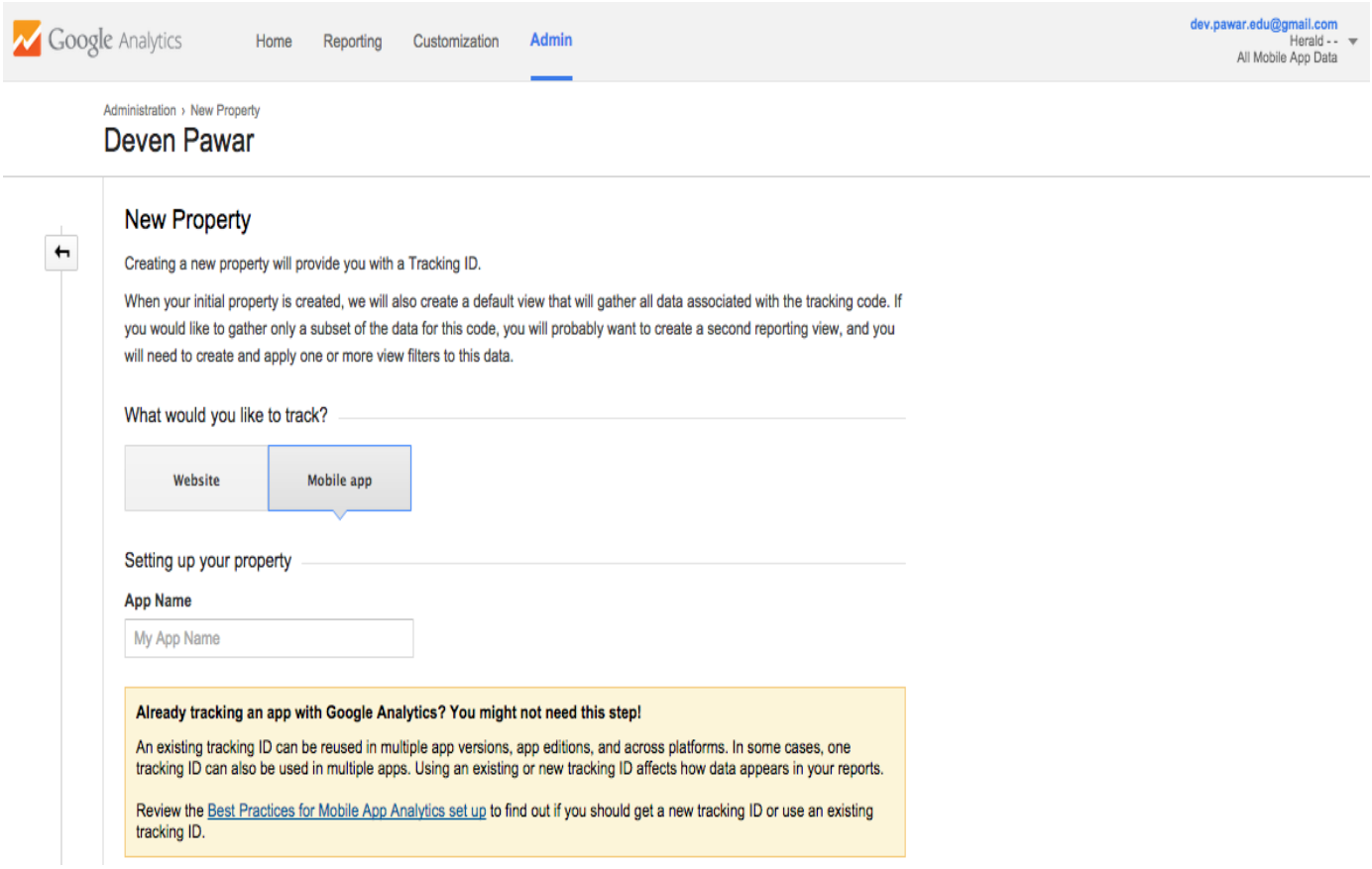
As the owner of the application we are always keen to know various usage statistics of our application such as how many times our app has been installed?, users from which part of the globe are accessing our app more frequently etc. To fulfill this requirement, we have used Google Analytics in our app.

Google Analytics is a service offered by Google for generating detailed statistics about the mobile or web application. The use of Google Analytics in our app has enabled following statistics of our applications:

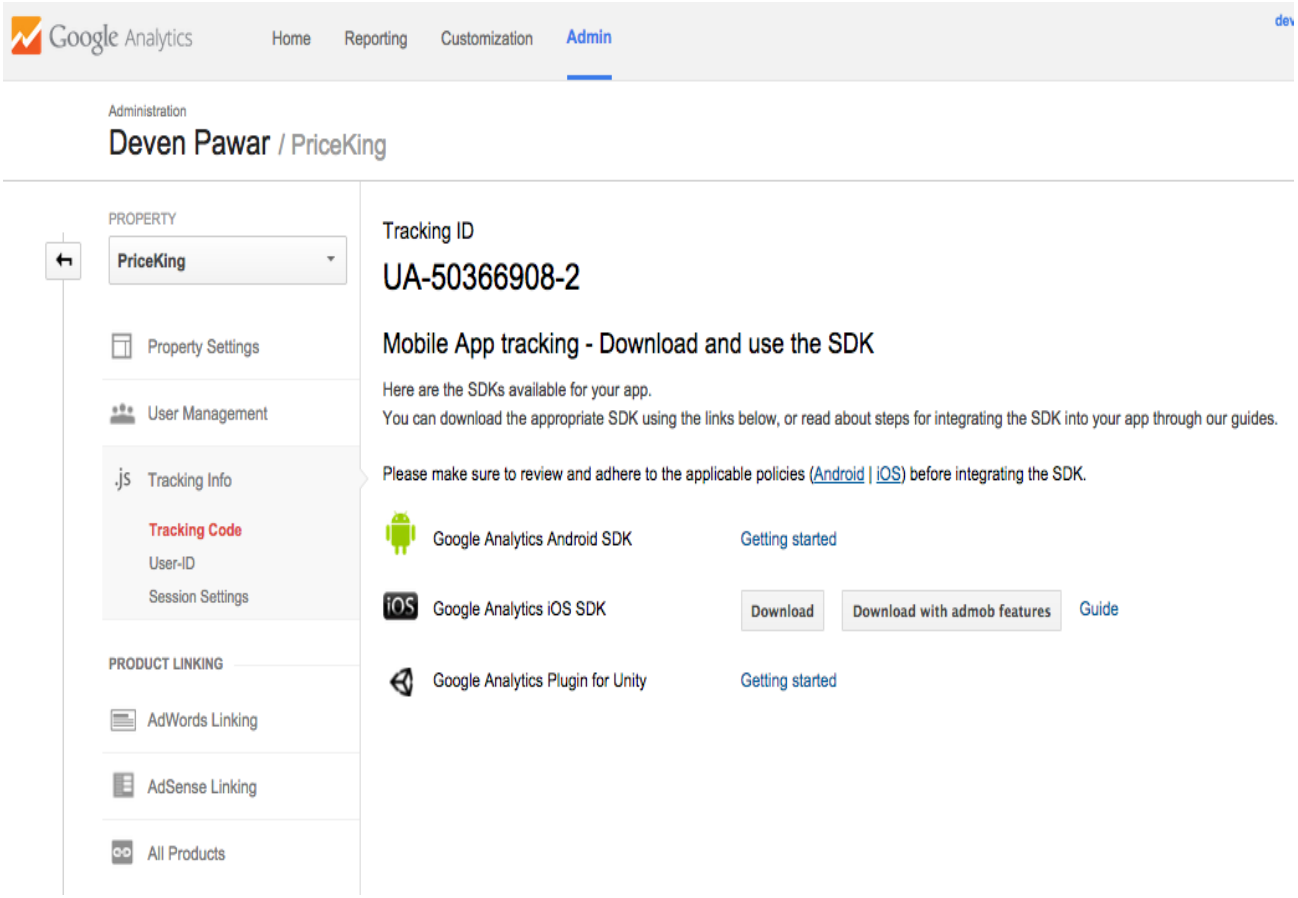
- App Installations
- Active users and demographics
- Screens and user engagements
- Crashes and exceptions

Create a new application/property under Google Analytics console.

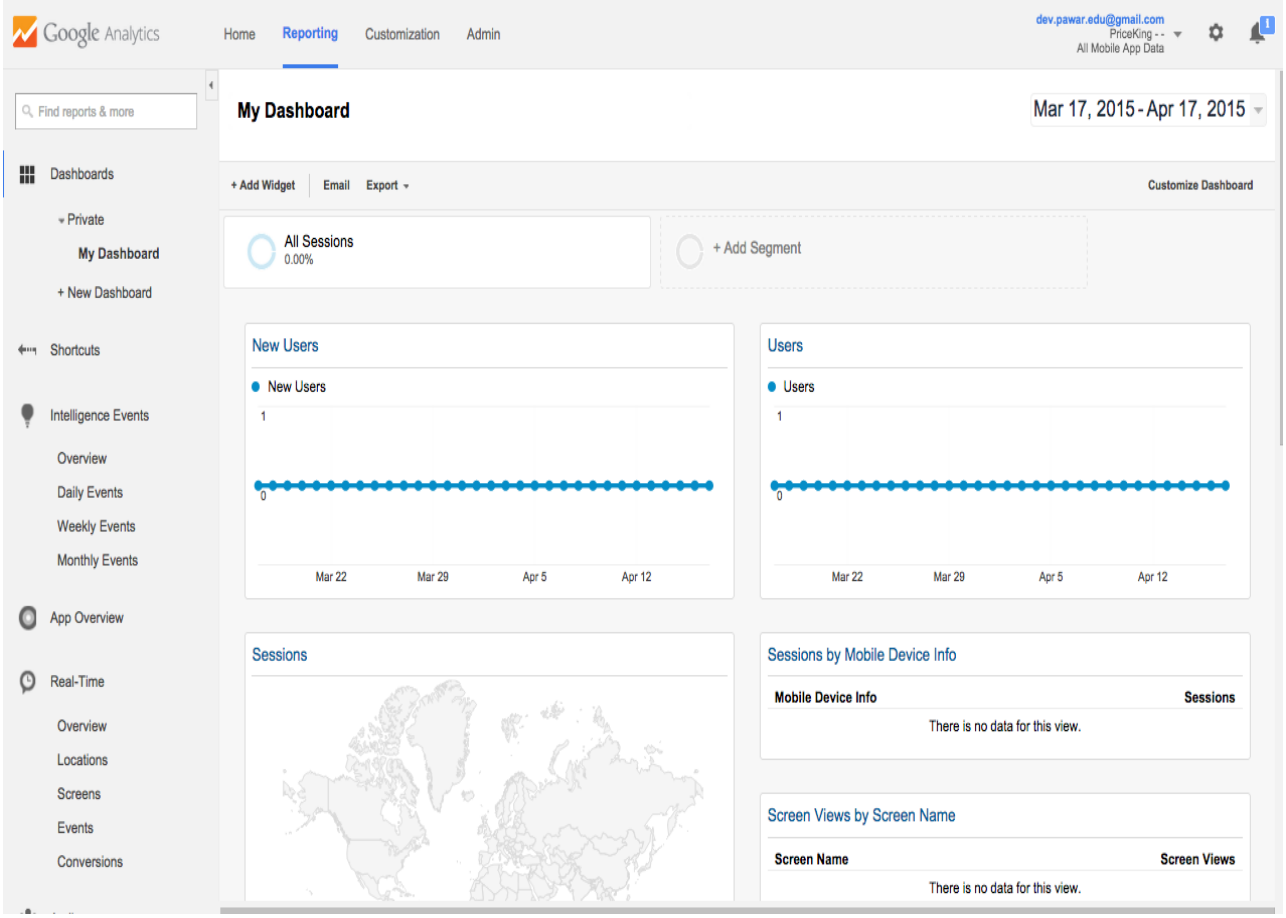
https://www.google.com/analytics/web/#management/Settings/a50366908w82333814p85252216/%3Fm.page%3DNewProperty/



This gives you the tracking id that you need to include in your Android application to perform google analytics.



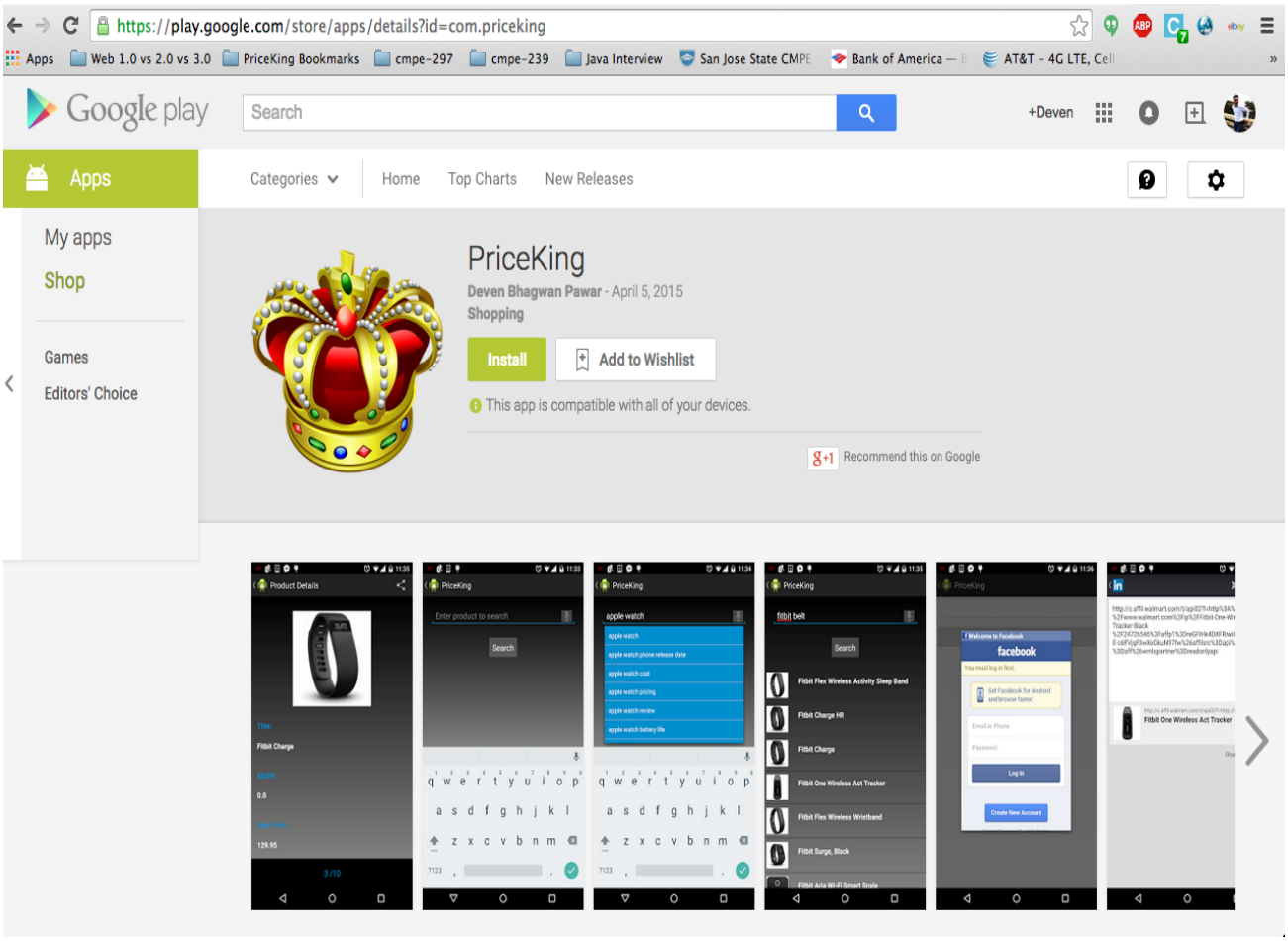
The following is the dashboard for PriceKing application where admin can see various aspect of the application explained above.



Cloud Deployment:

Google Play is the point of distribution of Android Apps. It allows all the users to view and download your Android applications previously known as Android Market. This is that one place where you can download your app from anywhere with the Internet connection enabled in your Android device. The PriceKing app is live on Google Play and users can download our app using the following link:

https://play.google.com/store/apps/details?id=com.priceking



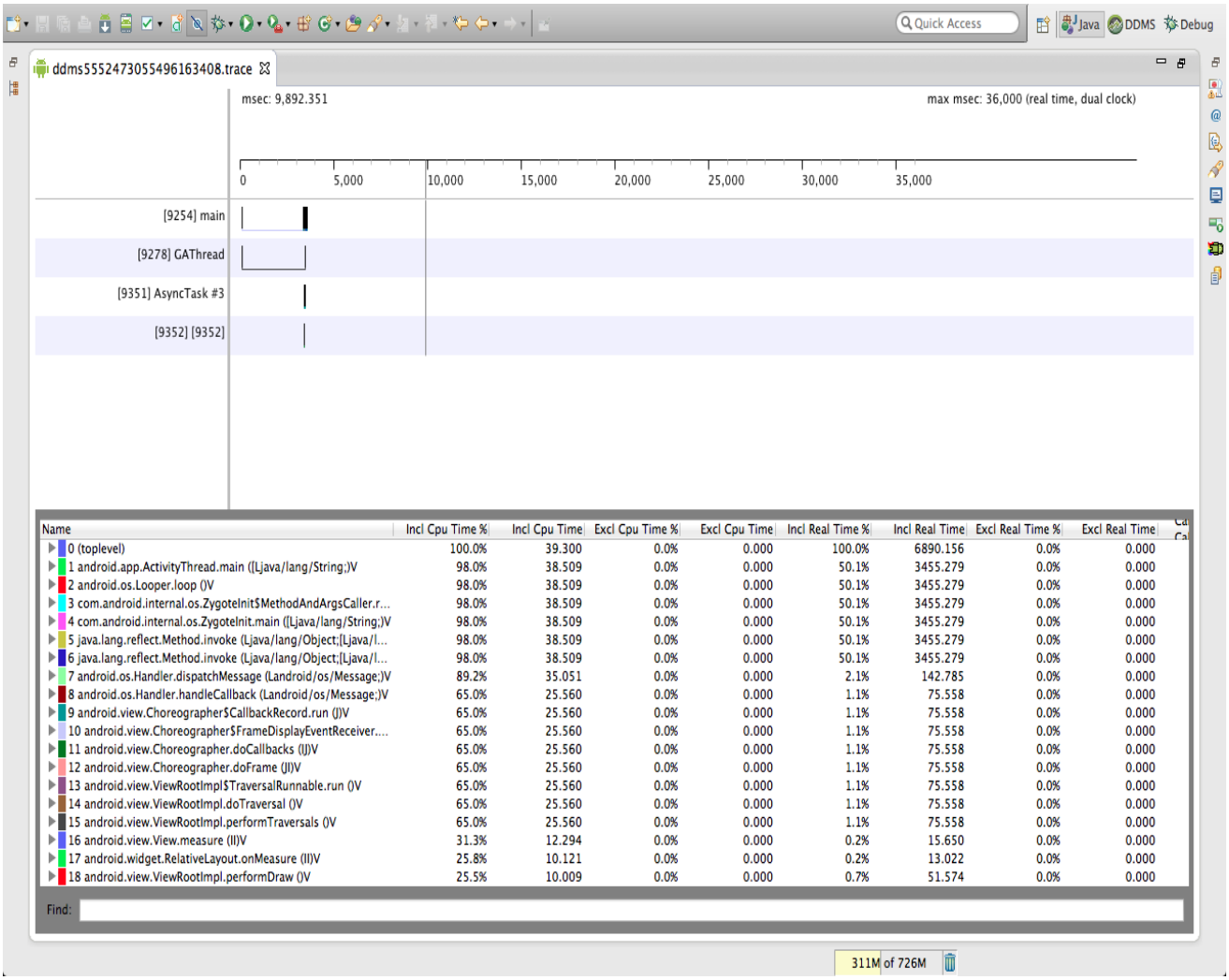
Profiling: Traceview is a graphical representation for execution logs that are developer generated using the class *debug*. Traceview helps in debugging the application and profiling is used for evaluating performance of the application.

Traceview Layout: Traceview displays the trace log data generated in DDMS view. The views can be described in the following two panels:

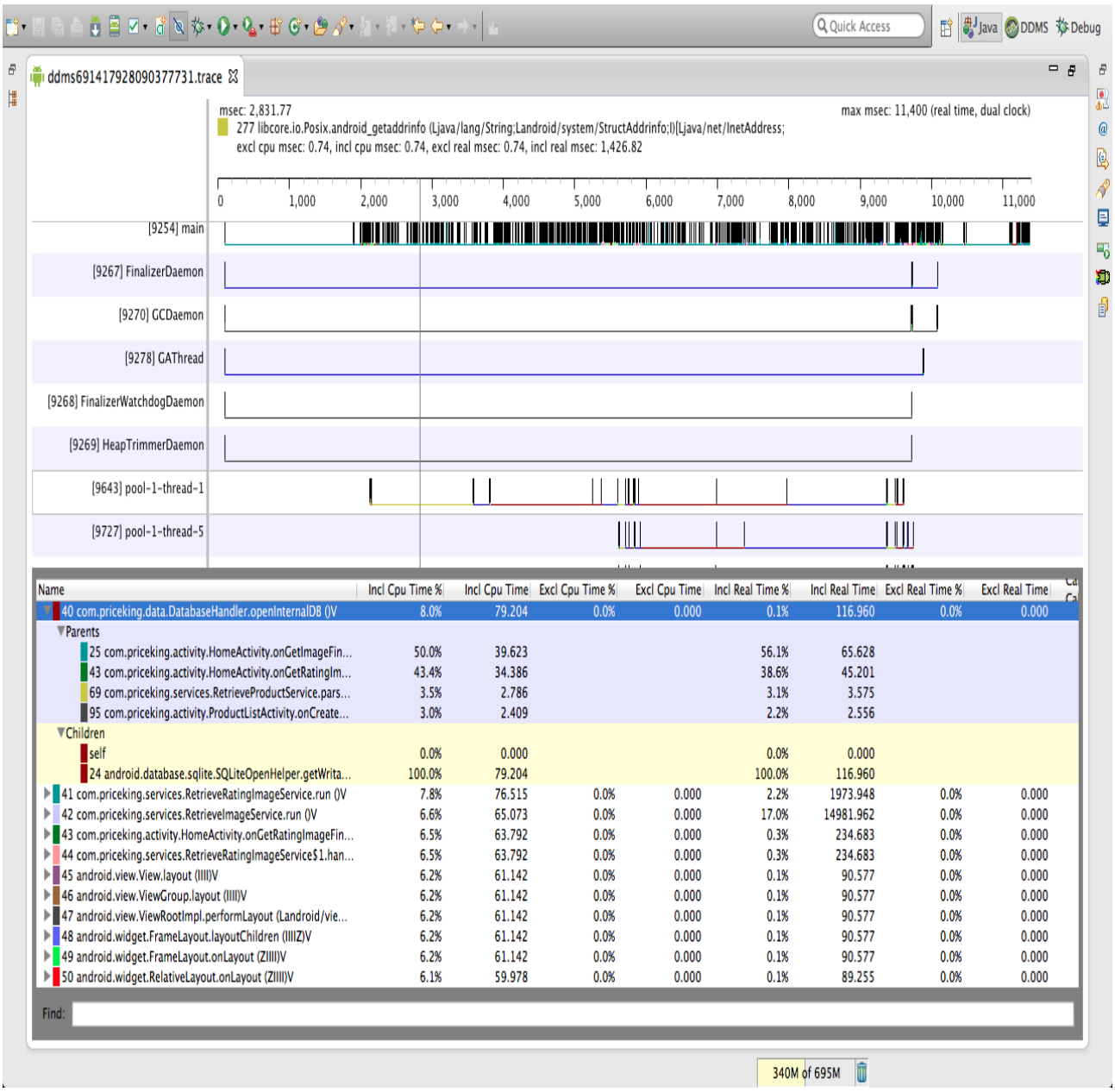
- A Timeline Panel – It shows when each thread and method started and stopped
- A Profile Panel – It provides a summary of execution of a method

Timeline Panel

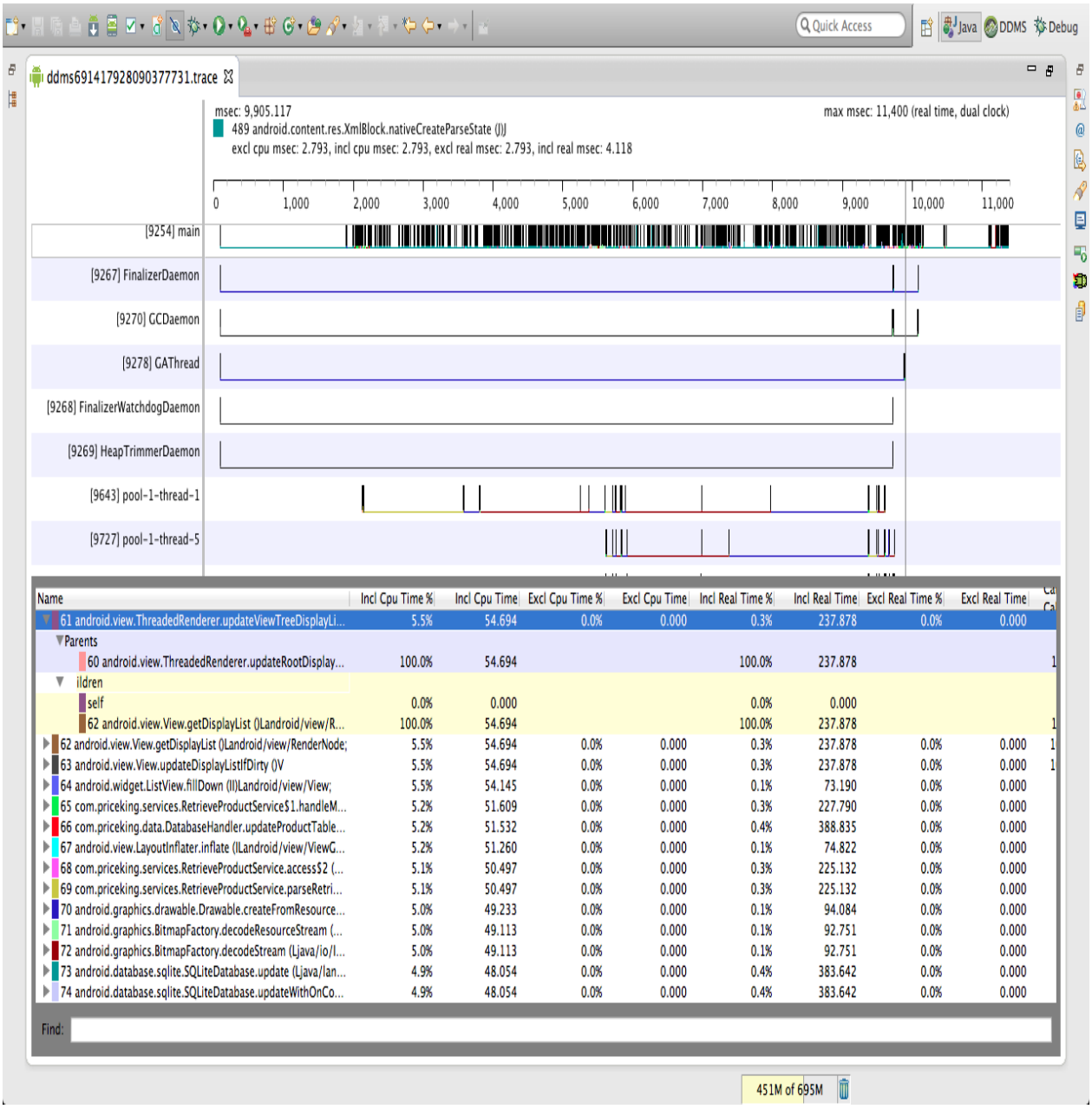
Timeline panel shows each thread's execution in its own row, with time progressing to the right. Every method is displayed in different color. The thin lines below the first row show the range of all the calls to the selected method.


Quick Access
Java
DDMS
Debug
ddms5552473055496163408.trace
msec: 9,892.351
max msec: 36,000 (real time, dual clock)
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
[9254] main
[9278] GAThread
[9351] AsyncTask #3
[9352] [9352]
Name | Incl Cpu Time % | Incl Cpu Time | Excl Cpu Time % | Excl Cpu Time | Incl Real Time % | Incl Real Time | Excl Real Time % | Excl Real Time
0 (toplevel) 100.0% 39.300 0.0% 0.000 100.0% 6890.156 0.0% 0.000
1 android.app.ActivityThread.main ([Ljava/lang/String;)V 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
2 android.os.Looper.loop ()V 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
3 com.android.internal.os.ZygoteInit$MethodAndArgsCaller.r... 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
4 com.android.internal.os.ZygoteInit.main ([Ljava/lang/String;)V 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
5 java.lang.reflect.Method.invoke (Ljava/lang/Object;[Ljava/l... 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
6 java.lang.reflect.Method.invoke (Ljava/lang/Object;[Ljava/l... 98.0% 38.509 0.0% 0.000 50.1% 3455.279 0.0% 0.000
7 android.os.Handler.dispatchMessage (Landroid/os/Message;)V 89.2% 35.051 0.0% 0.000 2.1% 142.785 0.0% 0.000
8 android.os.Handler.handleCallback (Landroid/os/Message;)V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
9 android.view.Choreographer$CallbackRecord.run (J)V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
10 android.view.Choreographer$FrameDisplayEventReceiver.... 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
11 android.view.Choreographer.doCallbacks (IJ)V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
12 android.view.Choreographer.doFrame (JI)V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
13 android.view.ViewRootImpl$TraversalRunnable.run ()V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
14 android.view.ViewRootImpl.doTraversal ()V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
15 android.view.ViewRootImpl.performTraversals ()V 65.0% 25.560 0.0% 0.000 1.1% 75.558 0.0% 0.000
16 android.view.View.measure (II)V 31.3% 12.294 0.0% 0.000 0.2% 15.650 0.0% 0.000
17 android.widget.RelativeLayout.onMeasure (II)V 25.8% 10.121 0.0% 0.000 0.2% 13.022 0.0% 0.000
18 android.view.ViewRootImpl.performDraw ()V 25.5% 10.009 0.0% 0.000 0.7% 51.574 0.0% 0.000
Find:
311M of 726M

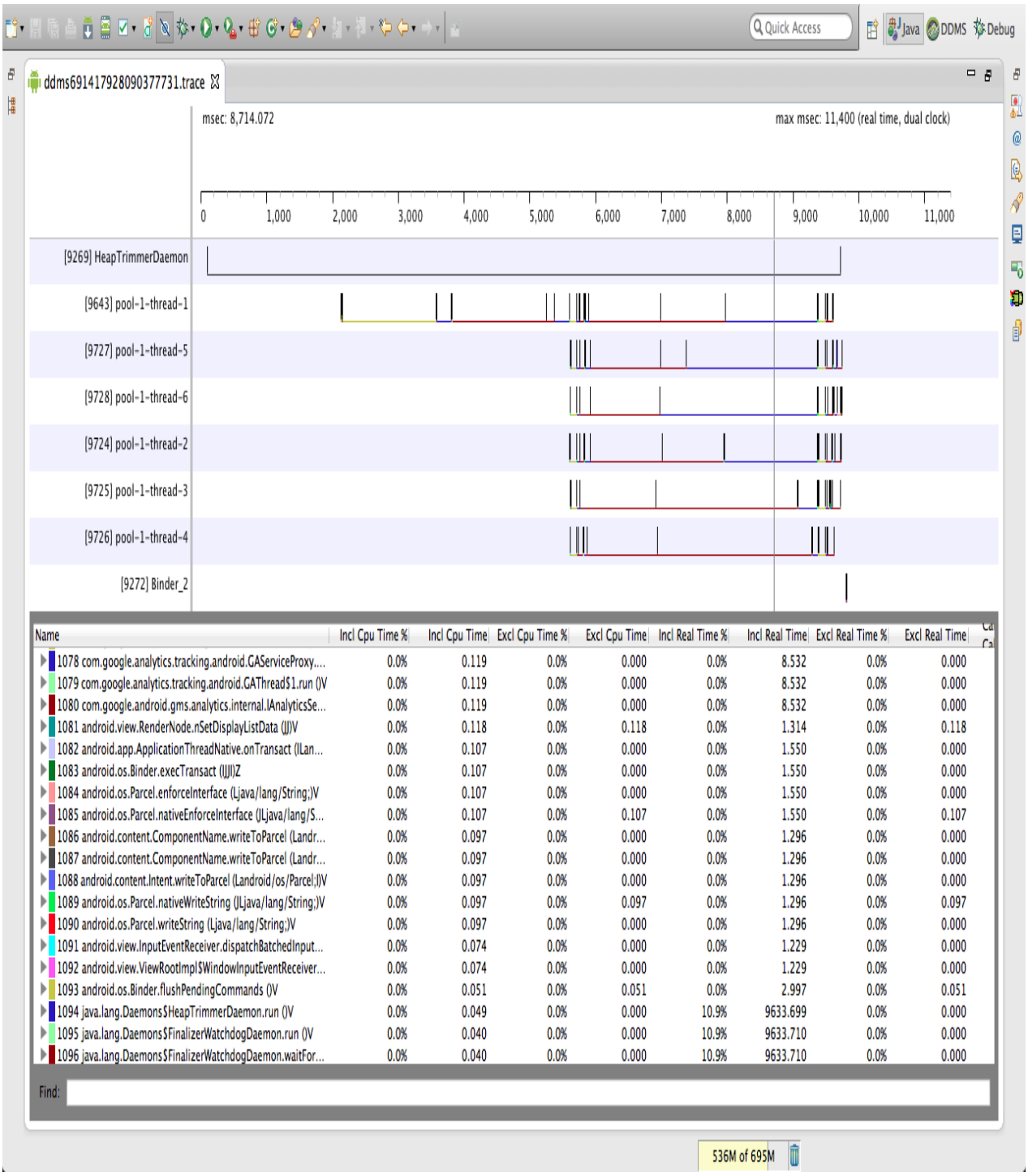
Traceview-1


Quick Access
Java
DDMS
Debug
ddms6914179280903777731.trace
msec: 2,831.77
max msec: 11,400 (real time, dual clock)
277 libcore.io.Posix.android_getaddrinfo (Ljava/lang/String;Landroid/system/StructAddrinfo;I)[Ljava/net/InetAddress;
excl cpu msec: 0.74, incl cpu msec: 0.74, excl real msec: 0.74, incl real msec: 1,426.82
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
[9254] main
[9267] FinalizerDaemon
[9270] GCDaemon
[9278] GAThread
[9268] FinalizerWatchdogDaemon
[9269] HeapTrimmerDaemon
[9643] pool-1-thread-1
[9727] pool-1-thread-5
Name
Incl Cpu Time %
Incl Cpu Time
Excl Cpu Time %
Excl Cpu Time
Incl Real Time %
Incl Real Time
Excl Real Time %
Excl Real Time
40 com.priceking.data.DatabaseHandler.openInternalDB ()V 8.0% 79.204 0.0% 0.000 0.1% 116.960 0.0% 0.000
Parents
25 com.priceking.activity.HomeActivity.onGetImageFin... 50.0% 39.623 56.1% 65.628
43 com.priceking.activity.HomeActivity.onGetRatingIm... 43.4% 34.386 38.6% 45.201
69 com.priceking.services.RetrieveProductService.pars... 3.5% 2.786 3.1% 3.575
95 com.priceking.activity.ProductListActivity.onCreate... 3.0% 2.409 2.2% 2.556
Children
self 0.0% 0.000 0.0% 0.000
24 android.database.sqlite.SQLiteOpenHelper.getWrita... 100.0% 79.204 100.0% 116.960
41 com.priceking.services.RetrieveRatingImageService.run ()V 7.8% 76.515 0.0% 0.000 2.2% 1973.948 0.0% 0.000
42 com.priceking.services.RetrieveImageService.run ()V 6.6% 65.073 0.0% 0.000 17.0% 14981.962 0.0% 0.000
43 com.priceking.activity.HomeActivity.onGetRatingImageFin... 6.5% 63.792 0.0% 0.000 0.3% 234.683 0.0% 0.000
44 com.priceking.services.RetrieveRatingImageService$1.han... 6.5% 63.792 0.0% 0.000 0.3% 234.683 0.0% 0.000
45 android.view.View.layout (IIII)V 6.2% 61.142 0.0% 0.000 0.1% 90.577 0.0% 0.000
46 android.view.ViewGroup.layout (IIII)V 6.2% 61.142 0.0% 0.000 0.1% 90.577 0.0% 0.000
47 android.view.ViewRootImpl.performLayout (Landroid/vie... 6.2% 61.142 0.0% 0.000 0.1% 90.577 0.0% 0.000
48 android.widget.FrameLayout.layoutChildren (IIIIZ)V 6.2% 61.142 0.0% 0.000 0.1% 90.577 0.0% 0.000
49 android.widget.FrameLayout.onLayout (ZIIII)V 6.2% 61.142 0.0% 0.000 0.1% 90.577 0.0% 0.000
50 android.widget.RelativeLayout.onLayout (ZIIII)V 6.1% 59.978 0.0% 0.000 0.1% 89.255 0.0% 0.000
Find:
340M of 695M

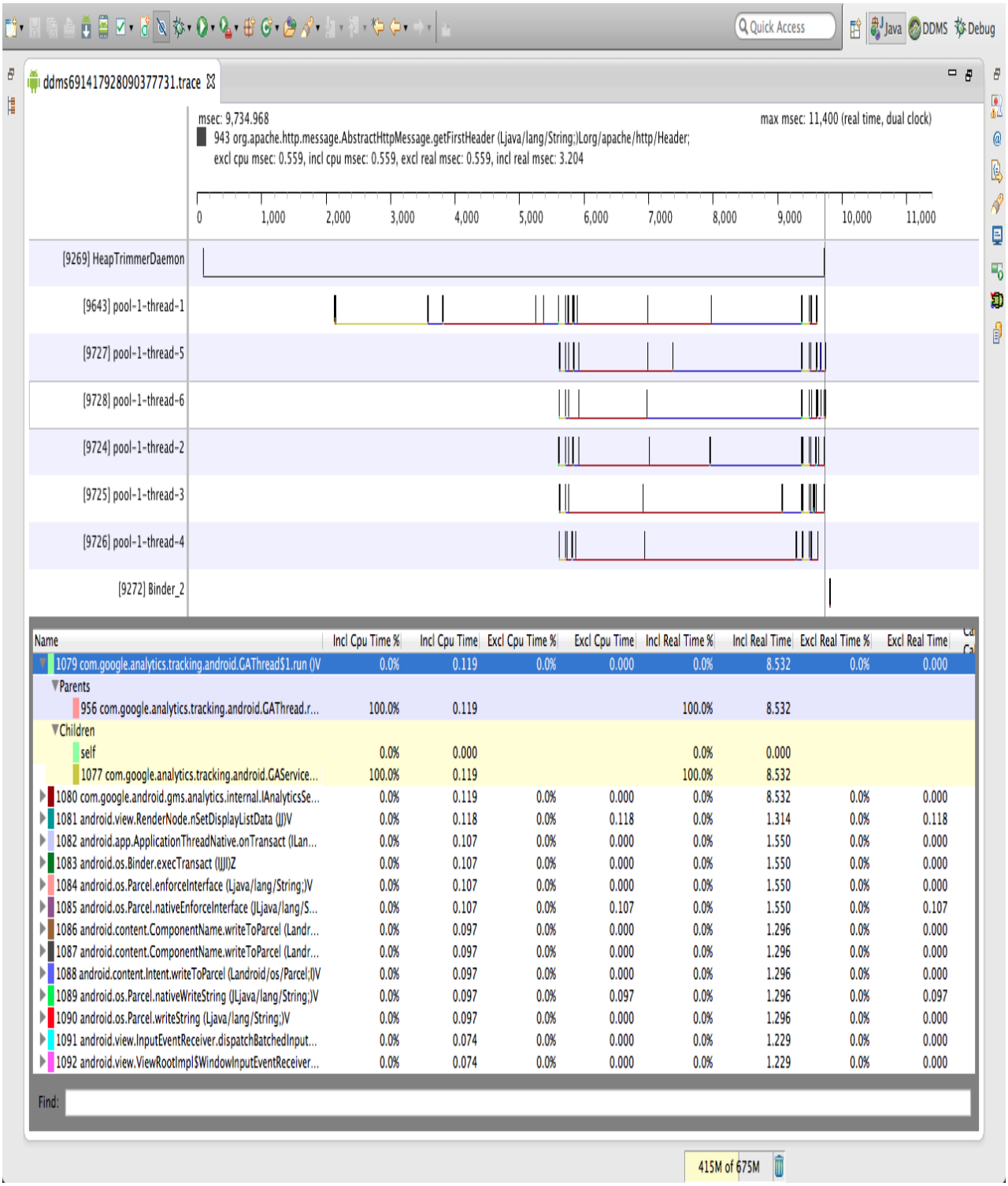
Traceview-2


Quick Access
Java
DDMS
Debug
ddms691417928090377731.trace
msec: 9,905.117
489 android.content.res.XmlBlock.nativeCreateParseState (JJ
excl cpu msec: 2.793, incl cpu msec: 2.793, excl real msec: 2.793, incl real msec: 4.118
max msec: 11,400 (real time, dual clock)
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
[9254] main
[9267] FinalizerDaemon
[9270] GCDaemon
[9278] GAThread
[9268] FinalizerWatchdogDaemon
[9269] HeapTrimmerDaemon
[9643] pool-1-thread-1
[9727] pool-1-thread-5
Name
Incl Cpu Time %
Incl Cpu Time
Excl Cpu Time %
Excl Cpu Time
Incl Real Time %
Incl Real Time
Excl Real Time %
Excl Real Time
61 android.view.ThreadedRenderer.updateViewTreeDisplayLi... 5.5% 54.694 0.0% 0.000 0.3% 237.878 0.0% 0.000
Parents
60 android.view.ThreadedRenderer.updateRootDisplay... 100.0% 54.694 100.0% 237.878
ildren
self 0.0% 0.000 0.0% 0.000
62 android.view.View.getDisplayList ()Landroid/view/R... 100.0% 54.694 100.0% 237.878
62 android.view.View.getDisplayList ()Landroid/view/RenderNode; 5.5% 54.694 0.0% 0.000 0.3% 237.878 0.0% 0.000
63 android.view.View.updateDisplayListIfDirty ()V 5.5% 54.694 0.0% 0.000 0.3% 237.878 0.0% 0.000
64 android.widget.ListView.fillDown (II)Landroid/view/View; 5.5% 54.145 0.0% 0.000 0.1% 73.190 0.0% 0.000
65 com.priceking.services.RetrieveProductService$1.handleM... 5.2% 51.609 0.0% 0.000 0.3% 227.790 0.0% 0.000
66 com.priceking.data.DatabaseHandler.updateProductTable... 5.2% 51.532 0.0% 0.000 0.4% 388.835 0.0% 0.000
67 android.view.LayoutInflater.inflate (ILandroid/view/ViewG... 5.2% 51.260 0.0% 0.000 0.1% 74.822 0.0% 0.000
68 com.priceking.services.RetrieveProductService.access$2 (... 5.1% 50.497 0.0% 0.000 0.3% 225.132 0.0% 0.000
69 com.priceking.services.RetrieveProductService.parseRetri... 5.1% 50.497 0.0% 0.000 0.3% 225.132 0.0% 0.000
70 android.graphics.drawable.Drawable.createFromResource... 5.0% 49.233 0.0% 0.000 0.1% 94.084 0.0% 0.000
71 android.graphics.BitmapFactory.decodeResourceStream (... 5.0% 49.113 0.0% 0.000 0.1% 92.751 0.0% 0.000
72 android.graphics.BitmapFactory.decodeStream (Ljava/io/I... 5.0% 49.113 0.0% 0.000 0.1% 92.751 0.0% 0.000
73 android.database.sqlite.SQLiteDatabase.update (Ljava/lan... 4.9% 48.054 0.0% 0.000 0.4% 383.642 0.0% 0.000
74 android.database.sqlite.SQLiteDatabase.updateWithOnCo... 4.9% 48.054 0.0% 0.000 0.4% 383.642 0.0% 0.000
Find:
451M of 695M

Traceview-3


Quick Access
Java
DDMS
Debug
ddms6914179280903777731.trace
msec: 8,714.072
max msec: 11,400 (real time, dual clock)
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
10,000
11,000
[9269] HeapTrimmerDaemon
[9643] pool-1-thread-1
[9727] pool-1-thread-5
[9728] pool-1-thread-6
[9724] pool-1-thread-2
[9725] pool-1-thread-3
[9726] pool-1-thread-4
[9272] Binder_2
Name
Incl Cpu Time %
Incl Cpu Time
Excl Cpu Time %
Excl Cpu Time
Incl Real Time %
Incl Real Time
Excl Real Time %
Excl Real Time
1078 com.google.analytics.tracking.android.GAServiceProxy.... 0.0% 0.119 0.0% 0.000 0.0% 8.532 0.0% 0.000
1079 com.google.analytics.tracking.android.GAThread$1.run ()V 0.0% 0.119 0.0% 0.000 0.0% 8.532 0.0% 0.000
1080 com.google.android.gms.analytics.internal.IAnalyticsSe... 0.0% 0.119 0.0% 0.000 0.0% 8.532 0.0% 0.000
1081 android.view.RenderNode.nSetDisplayListData (JJ)V 0.0% 0.118 0.0% 0.118 0.0% 1.314 0.0% 0.118
1082 android.app.ApplicationThreadNative.onTransact (ILan... 0.0% 0.107 0.0% 0.000 0.0% 1.550 0.0% 0.000
1083 android.os.Binder.execTransact (IJJI)Z 0.0% 0.107 0.0% 0.000 0.0% 1.550 0.0% 0.000
1084 android.os.Parcel.enforceInterface (Ljava/lang/String;)V 0.0% 0.107 0.0% 0.000 0.0% 1.550 0.0% 0.000
1085 android.os.Parcel.nativeEnforceInterface (JLjava/lang/S... 0.0% 0.107 0.0% 0.107 0.0% 1.550 0.0% 0.107
1086 android.content.ComponentName.writeToParcel (Landr... 0.0% 0.097 0.0% 0.000 0.0% 1.296 0.0% 0.000
1087 android.content.ComponentName.writeToParcel (Landr... 0.0% 0.097 0.0% 0.000 0.0% 1.296 0.0% 0.000
1088 android.content.Intent.writeToParcel (Landroid/os/Parcel;I)V 0.0% 0.097 0.0% 0.000 0.0% 1.296 0.0% 0.000
1089 android.os.Parcel.nativeWriteString (JLjava/lang/String;)V 0.0% 0.097 0.0% 0.097 0.0% 1.296 0.0% 0.097
1090 android.os.Parcel.writeString (Ljava/lang/String;)V 0.0% 0.097 0.0% 0.000 0.0% 1.296 0.0% 0.000
1091 android.view.InputEventReceiver.dispatchBatchedInput... 0.0% 0.074 0.0% 0.000 0.0% 1.229 0.0% 0.000
1092 android.view.ViewRootImpl$WindowInputEventReceiver... 0.0% 0.074 0.0% 0.000 0.0% 1.229 0.0% 0.000
1093 android.os.Binder.flushPendingCommands ()V 0.0% 0.051 0.0% 0.051 0.0% 2.997 0.0% 0.051
1094 java.lang.Daemons$HeapTrimmerDaemon.run ()V 0.0% 0.049 0.0% 0.000 10.9% 9633.699 0.0% 0.000
1095 java.lang.Daemons$FinalizerWatchdogDaemon.run ()V 0.0% 0.040 0.0% 0.000 10.9% 9633.710 0.0% 0.000
1096 java.lang.Daemons$FinalizerWatchdogDaemon.waitFor... 0.0% 0.040 0.0% 0.000 10.9% 9633.710 0.0% 0.000
Find:
536M of 695M

Traceview-4



Traceview-5

Profile Panel

The profile panel shows a summary of all the time spent in a method that includes both the inclusive and exclusive times. Exclusive time is the time elapsed in the method. Inclusive time is the time elapsed in the method and the time elapsed in any called functions. Calling methods are considered as "parents" whereas called methods are called as "children." Last column in the table shows the number of calls to the method and the number of recursive calls.

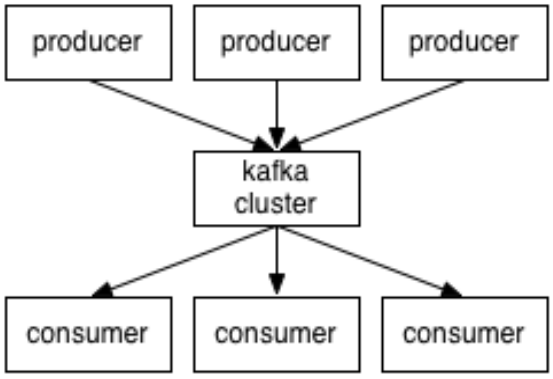
5.1.2 Platform

Platform services mainly consists of messaging and streaming api like kafka and storm and rest api calls for collecting the product deals from various vendors.

Apache Kafka:

Kafka is a distributed pub sub messaging system.Kafka is designed in such a way that even a single cluster can acts as the central data backbone for large organization.It can be elastically and transperently scaled without downtime. Data is partitioned and spread over clusters in order to process the data volume greater than a single machine can process.

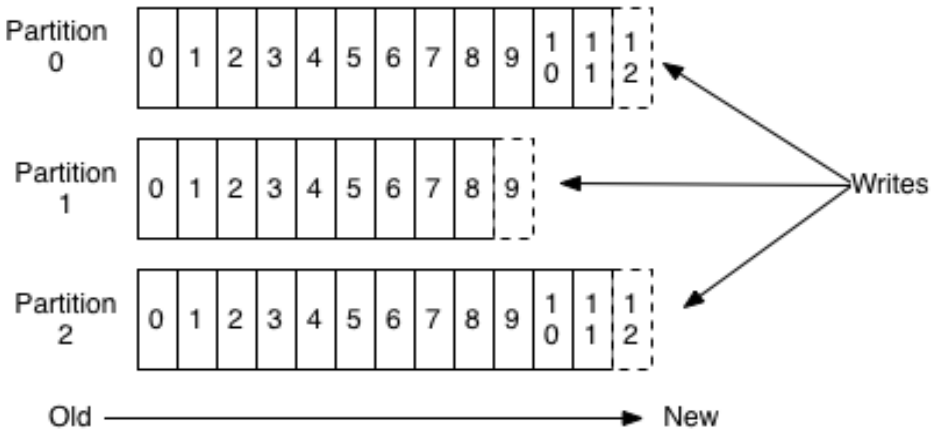
The high level architecture of kafka looks like this.



at high level producers send message to kafka cluster which in turn serves to the consumers by a high performance language agnostic TCP protocol.

Kafka Topic and logs

A topic is a feed to which the messages are published. Kafka maintains a partitioned log for each topic which is sequential immutable collection of messages where every new message is appended to the end of it and each message in the partition is uniquely identified by an offset.



Kafka messages are retained till the set retention policy even after the messages are consumed. This feature really differentiate kafka from other messaging systems when consumer can re-pull messages in case of failures.

The reason to go with kafka is first, consumer receives the messages produced by producer in order; second, for topic with N replication factor, kafka tolerates failures upto N-1 server failures without losing messages to log.

Dependencies to work with Kafka

In order to get kafka integrated to your system you need Zookeeper and zookeeper manages the kafka servers and let the producers and consumers about the changes in the kafka brokers.

Steps to run kafka:

1. start the zookeeper we run following command

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
```

2. Start the broker server

```
> bin/kafka-server-start.sh config/server.properties
```

3. create the topic

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test
```

4. Publish some messages

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
This is a message
This is another message
```

5. Test the consumer and run consumer

```
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning
This is a message
This is another message
```

Kafka Web console

Kafka web console is a open source utility to track your Kafka cluster on local as well as production cluster.

The web console project is a scala project built using play framework. So to use this utility in your project you need scala 2.9.2 installed on your server and also the latest version of play is required to run the project.

Web console also gives you functionality to add a zookeeper server on the fly and lets you monitor the brokers managed by that zookeeper server.

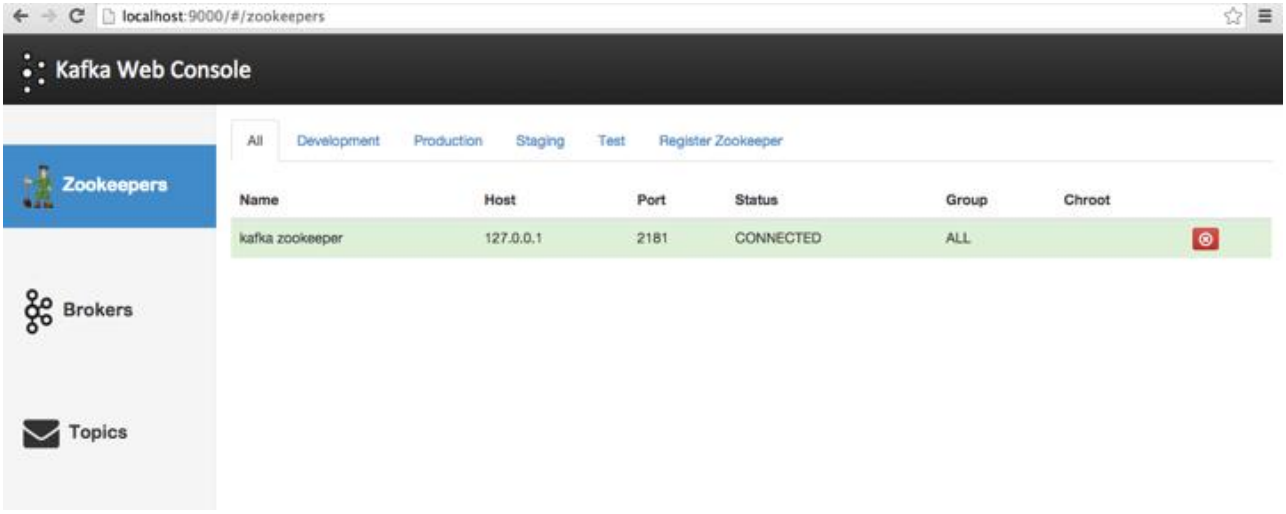
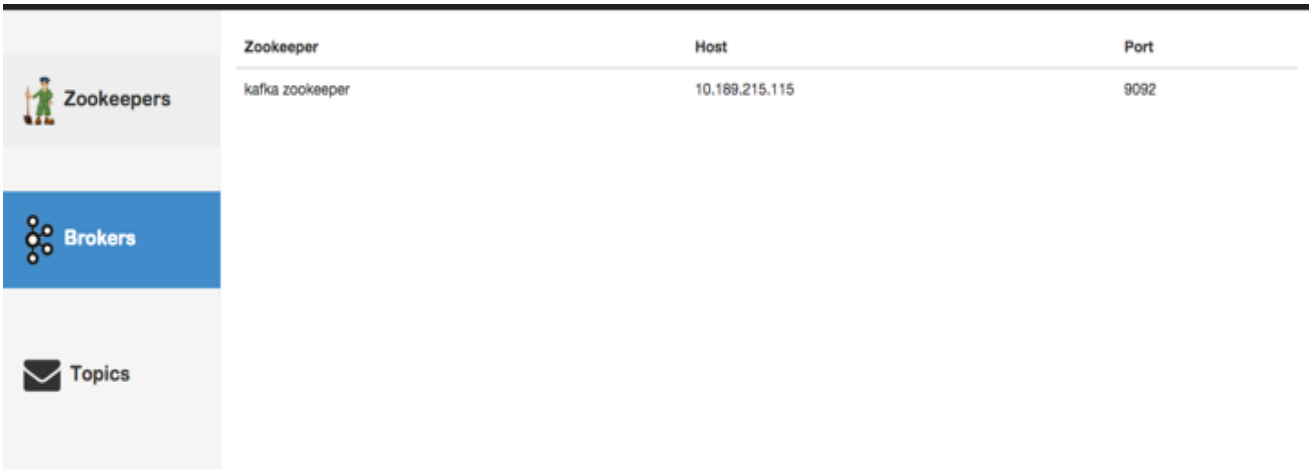
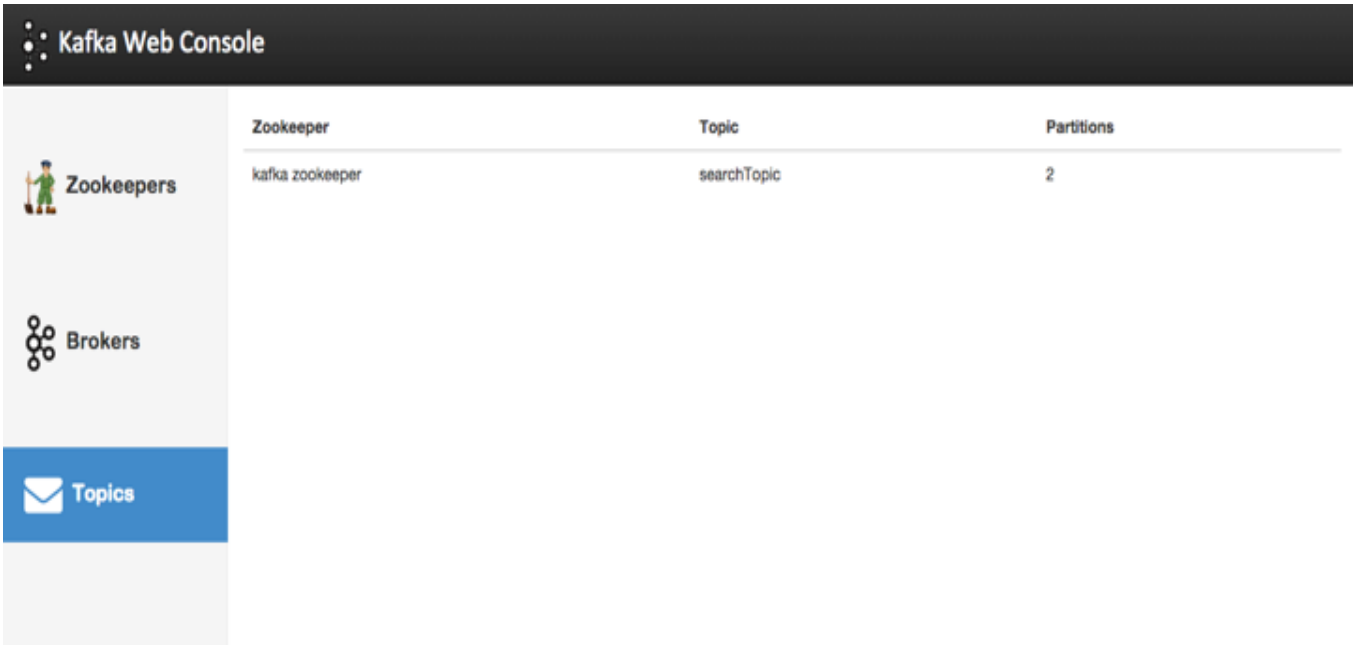


Figure 5.-





Not only this but it also lets user monitor the actual data flowing through your kafka topic

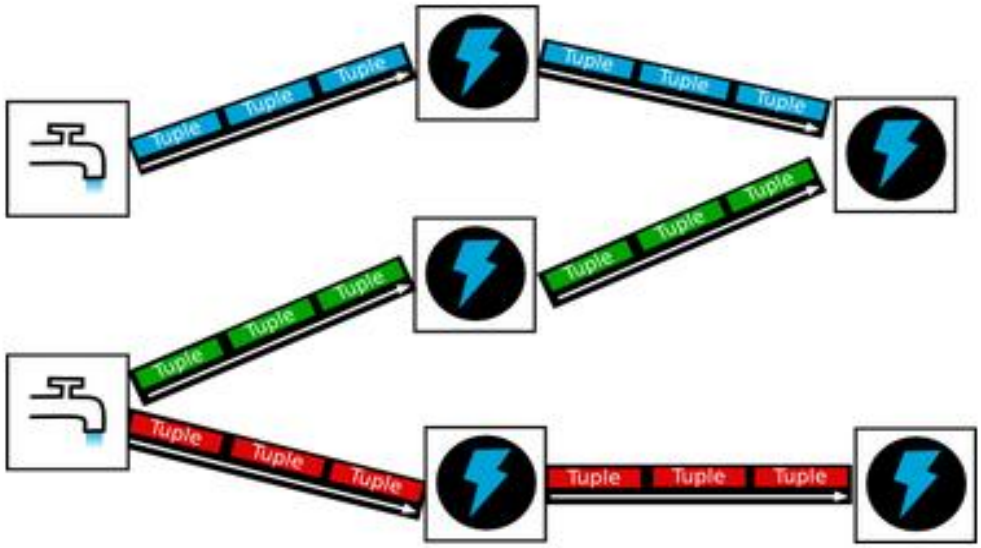
Steps to run the Kafka web console

```
Vinays-MacBook-Pro:~ vinaybhore$ cd /storage/ebay/kafka-web-console-1.1.0/
Vinays-MacBook-Pro:kafka-web-console-1.1.0 vinaybhore$ play start
[info] Loading project definition from /storage/ebay/kafka-web-console-1.1.0/pro
ject
[info] Set current project to kafka-web-console (in build file:/storage/ebay/kaf
ka-web-console-1.1.0/)

(Starting server. Type Ctrl+D to exit logs, the server will remain in background
)

Play server process ID is 3785
[info] play - database [default] connected at jdbc:h2:file:play
[info] play - Starting application default Akka system.
[info] play - Application started (Prod)
[info] play - Listening for HTTP on /0:0:0:0:0:0:0:0:9000
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeepe
r).
log4j:WARN Please initialize the log4j system properly.
```

Apache storm

Apache Storm is free and open source distributed real time computation system. Storm efficiently processes unbounded streams of data and processes it real time unlike hadoop which processes the job in batches. Storm is easy to use and can be used with any programming language.

The reason to choose Storm in our implementation is mainly its support for high stream of unbounded data, continuous computation and distributed RPC.

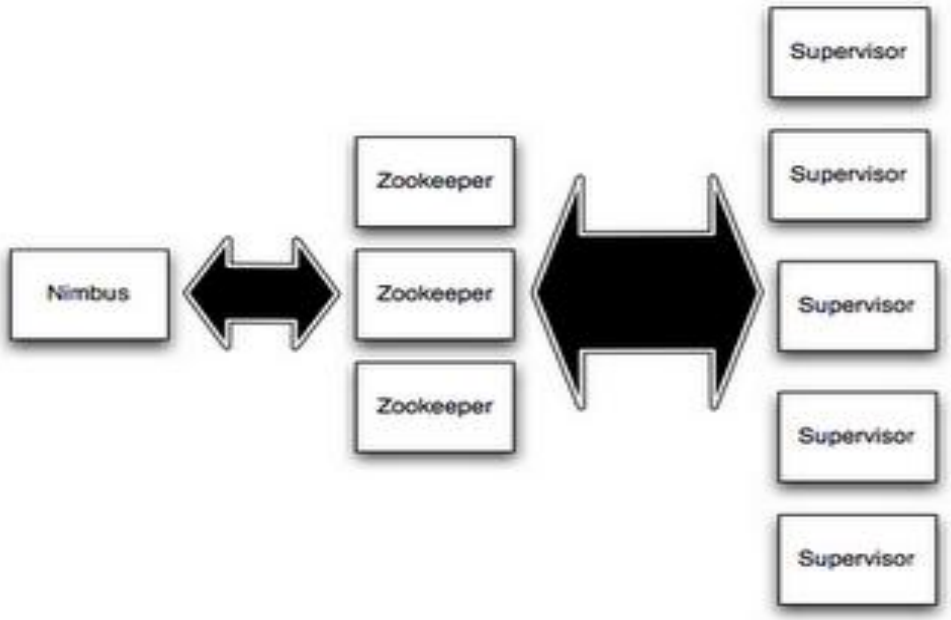


Storm Architecture

Storm cluster is similar to the hadoop cluster, hadoop runs mapreduce jobs whereas Storm runs topologies. The main difference between mapreduce job and a topology is mapreduce eventually finishes whereas the topology runs forever.

There are two kinds of nodes in Storm first Nimbus and other is Supervisor. MAster node runs a daemon called Nimbus which is similar to the hadoop job tracker. Nimbus distributes the code across the cluster assigning task to machines and tracks failures.

Worker node is called Supervisor. Supervisors does the work assigned to them by Nimbus. The communication between Nimbus and Supervisor is done through Zookeeper.



The idea of topology in our use case is to accept the tuple request from kafka and then push it to the bolts where each bolt is calling a single e-commerce api and get the results back in aggregator from each bolt.

The aggregated result is then filtered by esper bolt which is CEP and acts as a database upside down. In database we have a set of queries that we fire on the data when data is stored; here in esper we have queries stored and the data is flowing.

The demo topology is when ran has shown us the following output

Individual stages output in the topology

Stage 1

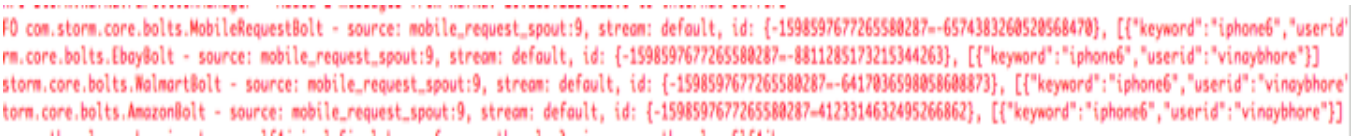
Spout receives a message from Kafka

Stage 2

Bolt picks up the message from spout and request is in next stage

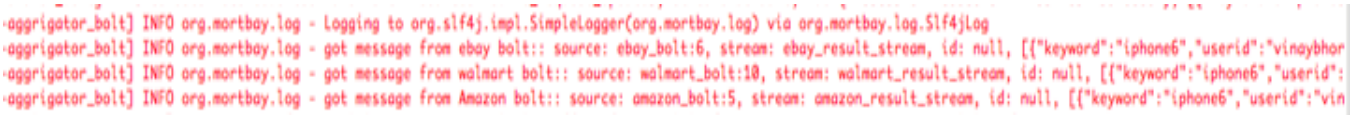
Stage 3

Parallel processing of the same request message across multiple e-commerce bolts



Stage 4

Aggregator received output from each of the e-commerce api



Topology Structure

The Storm topology is built by topology builder and the spouts and bolts are wired in this object.

```
public static StormTopology buildTopology() {

    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout(Constants.MOBILE_REQUEST_SPOUT_ID, new MobileRequestSpout(), 10);
    builder.setBolt(Constants.MOBILE_REQUEST_BOLT_ID, new MobileRequestBolt()).shuffleGrouping(Constants.MOBILE_REQUEST_SPOU
    builder.setBolt(Constants.AMAZON_BOLT_ID, new AmazonBolt(),1).shuffleGrouping(Constants.MOBILE_REQUEST_SPOUT_ID);
    builder.setBolt(Constants.EBAY_BOLT_ID, new EbayBolt(),1).shuffleGrouping(Constants.MOBILE_REQUEST_SPOUT_ID);
    builder.setBolt(Constants.WALMART_BOLT_ID, new WalmartBolt(),1).shuffleGrouping(Constants.MOBILE_REQUEST_SPOUT_ID);
    builder.setBolt(Constants.AGGRIGATOR_BOLT_ID, new AggrigatorBolt(),3).fieldsGrouping(Constants.AMAZON_BOLT_ID, Constants
```

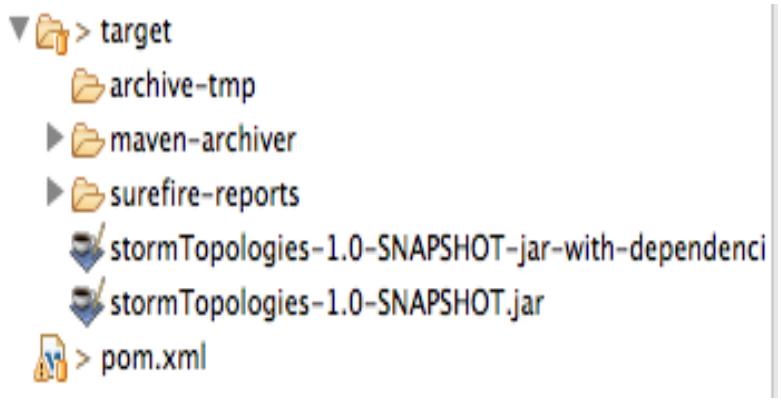
Now lets see how spout is configured

```
public class MobileRequestSpout extends KafkaSpout{


    private static final long serialVersionUID = 1L;
    public static SpoutConfig kafkaConfig = null;
    static{
        BrokerHosts brokerHosts = new ZkHosts("127.0.0.1");
        kafkaConfig = new SpoutConfig(brokerHosts, Constants.KAFKA_SEARCH_BUS, "", "uniqueIdentifier");
        kafkaConfig.scheme = new SchemeAsMultiScheme(new StringScheme());
        List<String>zkList = new ArrayList<String>();
        zkList.add("localhost");
        kafkaConfig.zkPort= 2181;
        kafkaConfig.zkServers = zkList;

    }

    public MobileRequestSpout() {
        super(kafkaConfig);

    }



}
```

Bolts are configured in the following way

```
public class AmazonBolt extends BaseRichBolt{

    /**
     *
     */
    private OutputCollector collector;
    public static final Logger LOG = LoggerFactory.getLogger(AmazonBolt.class);
    public AmazonBolt() {
        // TODO Auto-generated constructor stub
    }

    public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
        // TODO Auto-generated method stub
        this.collector =collector;
    }

    public void execute(Tuple input) {
        // TODO Auto-generated method stub
        this.collector.emit(Constants.AMAZONRESULTSTREAM,new Values(input.getValue(0)));
        LOG.info(input.toString());

    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        // TODO Auto-generated method stub
        declarer.declareStream(Constants.AMAZONRESULTSTREAM, new Fields("rs"));

    }
```

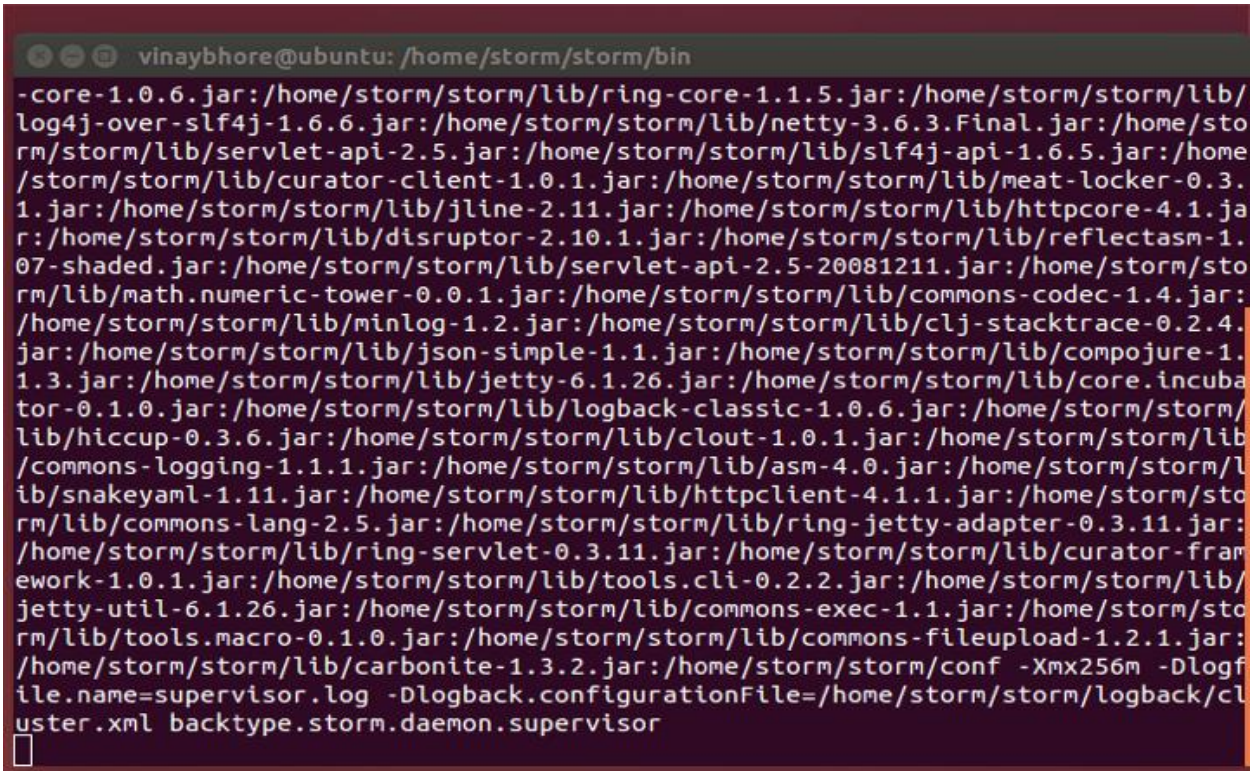
The topology has to run on server continuously so we have to package the Storm topolgy in a fat jar with dependencies so we add a build plug-in in our pom and generate a jar with dependencies under target folder in our project workspace.

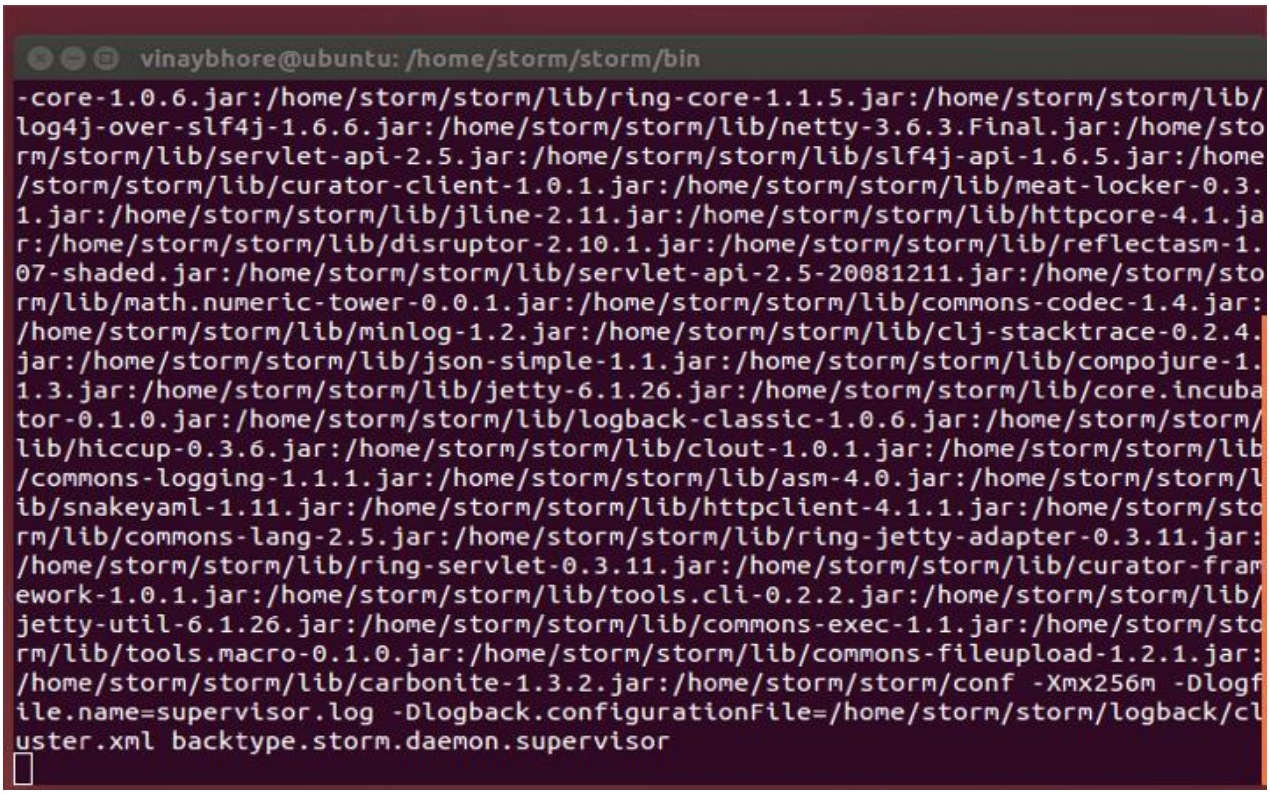



```
<plugins>
    <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
            <archive>
                <manifest>
                    <mainClass>com.storm.core.SearchTopology</mainClass>
                </manifest>
            </archive>
            <descriptorRefs>
                <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
        </configuration>
        <executions>
            <execution>
                <id>make-assembly</id> <!-- this is used for inheritance me
                <phase>package</phase> <!-- bind to the packaging phase -->
                <goals>
                    <goal>single</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
```

Step 1

Before running the Storm jar on cloud or cluster start the Nimbus and Supervisor





Step 2

Run storm UI to monitor and control topology on UI would be available on port 8772

```
vinaybhore@ubuntu: /home/storm/storm/bin
/storm/storm/lib/curator-client-1.0.1.jar:/home/storm/storm/lib/meat-locker-0.3.
1.jar:/home/storm/storm/lib/jline-2.11.jar:/home/storm/storm/lib/httpcore-4.1.ja
r:/home/storm/storm/lib/disruptor-2.10.1.jar:/home/storm/storm/lib/reflectasm-1.
07-shaded.jar:/home/storm/storm/lib/servlet-api-2.5-20081211.jar:/home/storm/sto
rm/lib/math.numeric-tower-0.0.1.jar:/home/storm/storm/lib/commons-codec-1.4.jar:
/home/storm/storm/lib/minlog-1.2.jar:/home/storm/storm/lib/clj-stacktrace-0.2.4.
jar:/home/storm/storm/lib/json-simple-1.1.jar:/home/storm/storm/lib/compojure-1.
1.3.jar:/home/storm/storm/lib/jetty-6.1.26.jar:/home/storm/storm/lib/core.incuba
tor-0.1.0.jar:/home/storm/storm/lib/logback-classic-1.0.6.jar:/home/storm/storm/
lib/hiccup-0.3.6.jar:/home/storm/storm/lib/clout-1.0.1.jar:/home/storm/storm/lib
/commons-logging-1.1.1.jar:/home/storm/storm/lib/asm-4.0.jar:/home/storm/storm/l
ib/snakeyaml-1.11.jar:/home/storm/storm/lib/httpclient-4.1.1.jar:/home/storm/sto
rm/lib/commons-lang-2.5.jar:/home/storm/storm/lib/ring-jetty-adapter-0.3.11.jar:
/home/storm/storm/lib/ring-servlet-0.3.11.jar:/home/storm/storm/lib/curator-fram
ework-1.0.1.jar:/home/storm/storm/lib/tools.cli-0.2.2.jar:/home/storm/storm/lib/
jetty-util-6.1.26.jar:/home/storm/storm/lib/commons-exec-1.1.jar:/home/storm/sto
rm/lib/tools.macro-0.1.0.jar:/home/storm/storm/lib/commons-fileupload-1.2.1.jar:
/home/storm/storm/lib/carbonite-1.3.2.jar:/home/storm/storm:/home/storm/storm/co
nf -Xmx768m -Dlogfile.name=ui.log -Dlogback.configurationFile=/home/storm/storm/
logback/cluster.xml backtype.storm.ui.core
```

Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.1-incubating	24s	0	0	0	0	0	0

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks

Supervisor summary

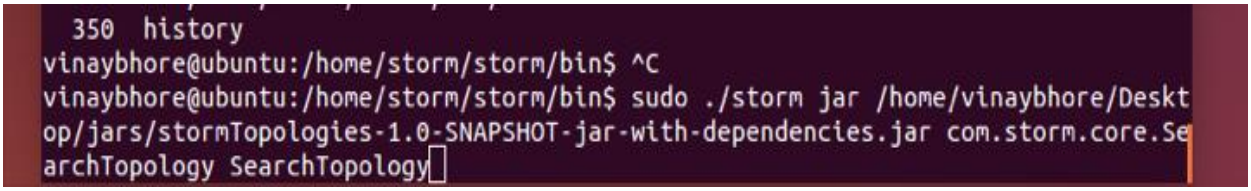
Id	Host	Uptime	Slots	Used slots

Nimbus Configuration

Key	Value
dev.zookeeper.path	/tmp/dev-storm-zookeeper
drpc.childopts	-Xmx768m
drpc.invocations.port	3773
drpc.port	3772
drpc.queue.size	128
drpc.request.timeout.secs	600
drpc.worker.threads	64
java.library.path	/usr/lib/jvm/java-6-openjdk-amd64

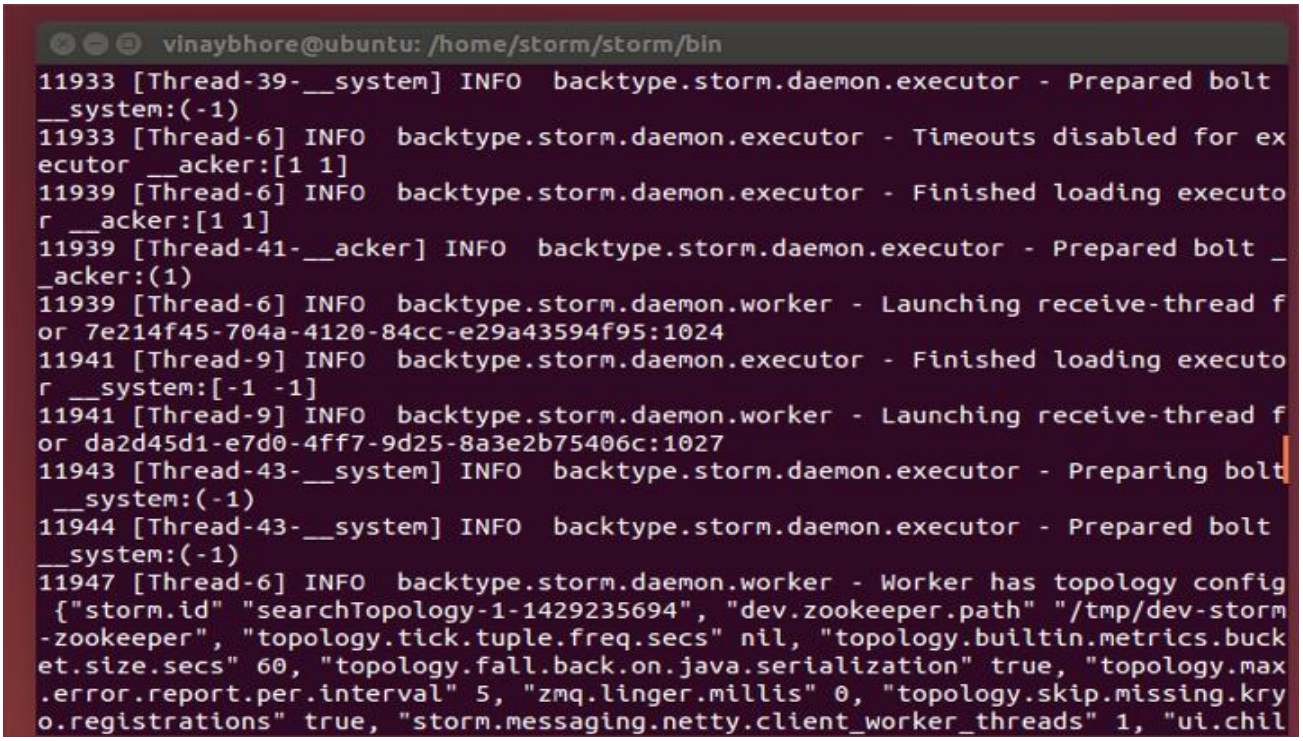
Step 3

On the storm cluster we run the topology jar with dependencies using the following command



Step 4

Monitor the topology on console, it looks like this



5.2 System implementation issues and resolutions

5.2.1 Implementation issues in Platform services

Setting up the environment on different platform was the biggest challenge. We had faced many version issues and dependency conflicts while setting up our clusters and deployment of our topologies on single node Storm cluster and finally on cloud deployment.

Following are the issues we have faced while developing setting up and deploying our topologies.

1. Kafka binary comes with the default configurations in server.properties which was causing our data to get accumulated and was not deleted and causing our hard drives to fill out when the topology ran for few hours.

Solution to that is the log retention policy to set up for hourly basis when we were testing on our local environment.

2. Kafka binary version 8.0 does have conflicts with Storm binary and creates problem in integration

Solution to this is using the kafka binary version 8.1.1 with scala sbt version to 2.9.2

3. Kafka Web console was not running play project

We fixed it by updating sbt and installing latest play framework

4. Storm 9.3 binary was creating issue in deployment with our jar

It was not recognizing the storm -kafka integration jar used from one of the contrib projects when Storm was under incubation.

Solved this issue by building and installing Storm 9.0.1 version from source by installing zookeeper, zeromq, and Jzmq and Storm 9.0.1 incubation project by Nathan Marz on github.

5. jzmq was not building

solved this issue by updating configure.sh file after building the project and providing the necessary dependencies in the file.

6. Storm workers were dying

Added the tick time while building the topology so that topology remains live after initial setup.

7. Deployment was not recognizing the zookeeper

Solved this issue using single zookeeper instance for both kafka and storm on the same cluster.

Issues	Resolution
Integration	Individual parts were working. Eg - Kafka-Storm, UI, Web Application, Mobile App were running fine. But integration broke the build.
OCR Implementation	OCR implementation source code is available for android on github (rmtheis) but there were many errors in the code. Also, the OCR was not working properly. It could not detect the image properly. Response time was the issue. We used other libraries to resolve this. Moreover, the

	mistakes in the code needed to be solved.
API product/coupon format	JSON format of the products and coupons were different for all the APIs. Challenge was to bring all the data in common format before filtering it.

Table 5.1: Implementation issues and Resolutions

5.3 Used technologies and tools

5.3.1Used Tools:

1. Apache Kafka Web console- Kafka is an open-source message broker written in scala. It provides a unified, high-throughput, low-latency platform for handling real-time data feeds.
2. Storm UI - Storm is a free and open source distributed realtime computation system. It makes it easy to reliably process unbounded streams of data, doing for realtime processing. It can be used with any programming language.
3. AWS - It is a collection of remote computing services that together make a cloud computing platform.
4. MongoVUE - It is an innovative MongoDB desktop application that gives an elegant and highly usable user interface to work with MongoDB.
5. Github - Online project hosting using Git. It is free for public open-source code and includes source-code browser, in-line editing, wikis, etc.
6. Testdroid- It offers agile mobile app test automation on ANdroid and iOS devices.
7. Selenium webdriver - Used for Web Application testing.
8. Android SDK - Development tool used for building android apps.
9. Eclipse IDE - It is a Java development tool.

5.3.2 Adopted Standards:

Standard Name	Engineering Process	Note

Unified software development process	Project Management	Based on the platform on which the application and program is going to be designed, we plan to use test driven development technique. We will take bottom up approach as the test cases will be written first
Javadoc 1.7	Source documentation	We are using Eclipse IDE as it has the native support. Source of other required tools will be Oracle and Apache website
IEEE/ APA citation reference	Report documentation	
Java EE 6 SDK	Back-end application development	

Table 5.2: Adopted Standards

6. System Testing and Experiment

6.1 Testing and Experiment Scope

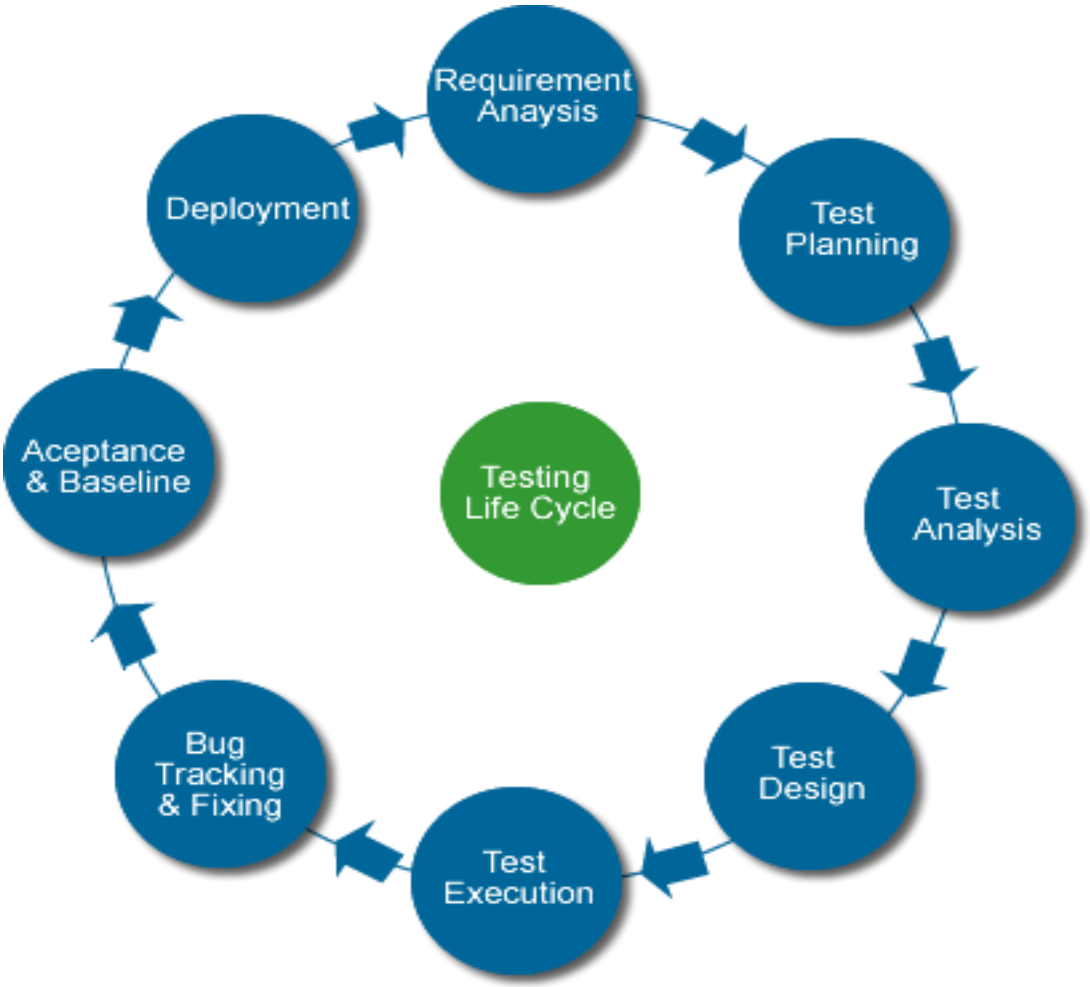


Figure 6.1: Testing Lifecycle

Below are the few test cases that we have framed.

Test Case ID	Test Case Description	Expected Result	Actual Result (Pass/ Fail)
1.	Test the client side services and storage for specific deal search and sort based on user selected criteria.	It should search best deal from all the available deals.	

2.	User profile CRUD operation testing	User profile is created, read, updated, deleted.	
3.	Coupon search testing	Coupons are searched for the particular search.	
4.	All the mobile and tablet platforms are supported	Application should run on all the platforms.	
5.	Authentication of desired category of users viz. admin, small business, etc	Users should be authenticated based on category.	
6.	Mobile Application UI testing	All the mobile User Interfaces should be well formated.	
7.	Dashboard UI testing	All the web User Interfaces should be well formated.	
8.	Location based analytics	Analytics is performed based on the location.	
9.	Test the feature of sharing of deals	User is able to share deals.	
10.	Test that the user's searching pattern is saved	User's searching pattern is saved or not.	
11.	Test that the advertisements are displayed based on user profile	Advertisements are shown based on user's interest.	
15.	Email notification testing	Email is sent to user on regular intervals.	

16.	Coupons CRUD operation testing	Coupon is created, read, updated, deleted.	
17.	Product CRUD operation testing	Product is created, read, updated, deleted.	

Table 7.1: Adopted Standards

6.2 Testing and Experimental Approaches

There are several platforms for mobile app UI and functional testing. Below are the tools that we are planning to use for testing.

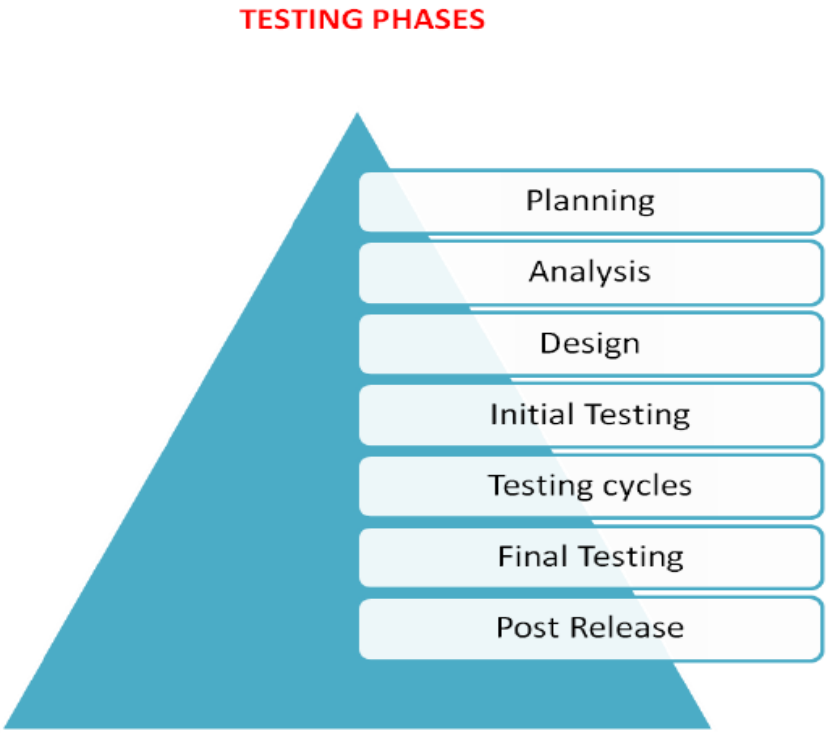


Figure 6.2 Testing Phases

Selenium WebDriver:

Selenium is an open-source portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests. It supports multiple languages and platforms. It can be downloaded and used without charge.

TestDroid:

Testdroid is a set of mobile software development and testing product. It provides an API through open source software available on github. It can use testing frameworks such as Robotium and Appium. Testdroid Recorder can also be used which is available at Eclipse marketplace.

6.2.1 Agile test approach

Development as well as testing approach was incremental and iterative as in Agile Test methodology . Strategy followed for testing as well as development is as shown below -

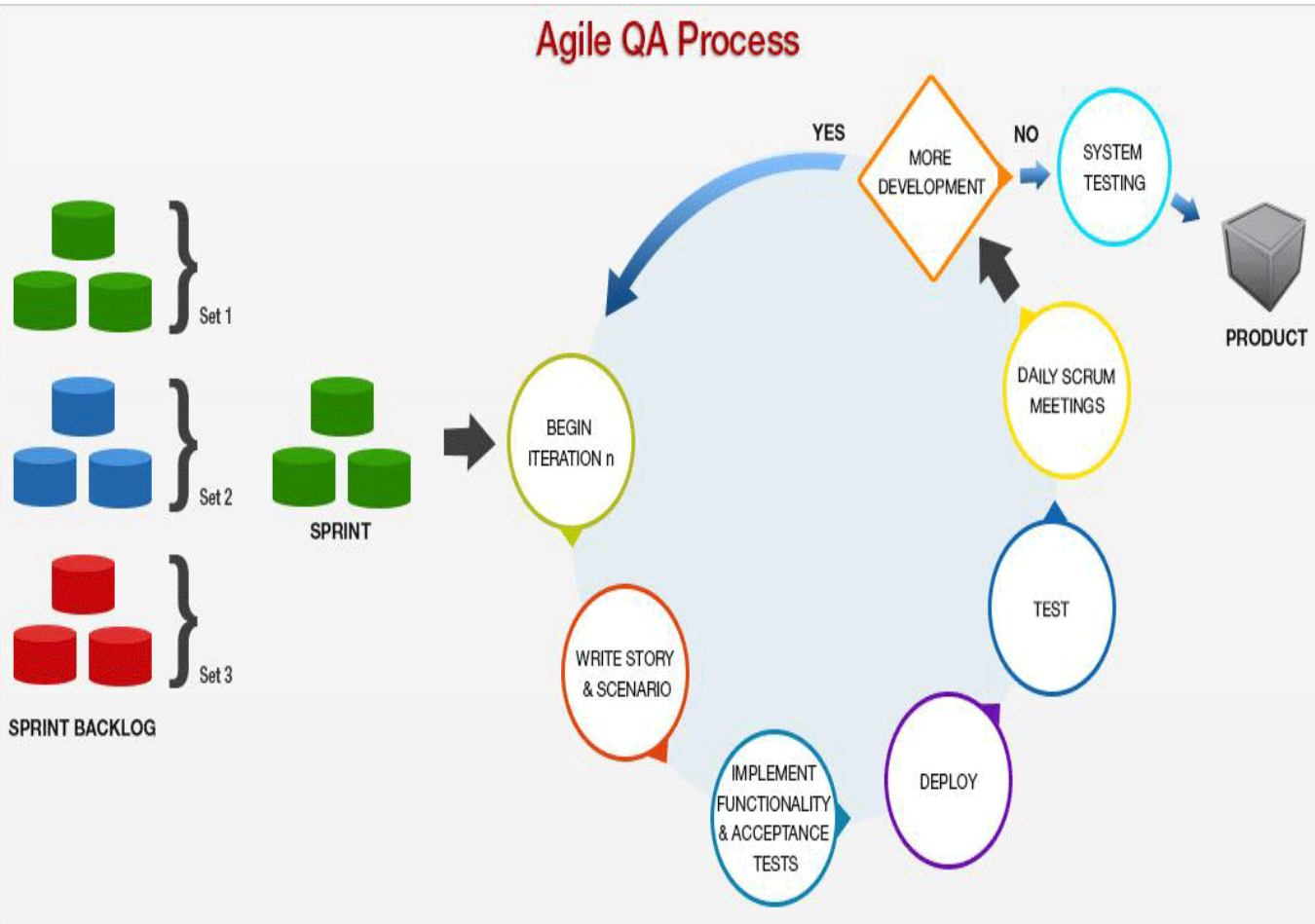
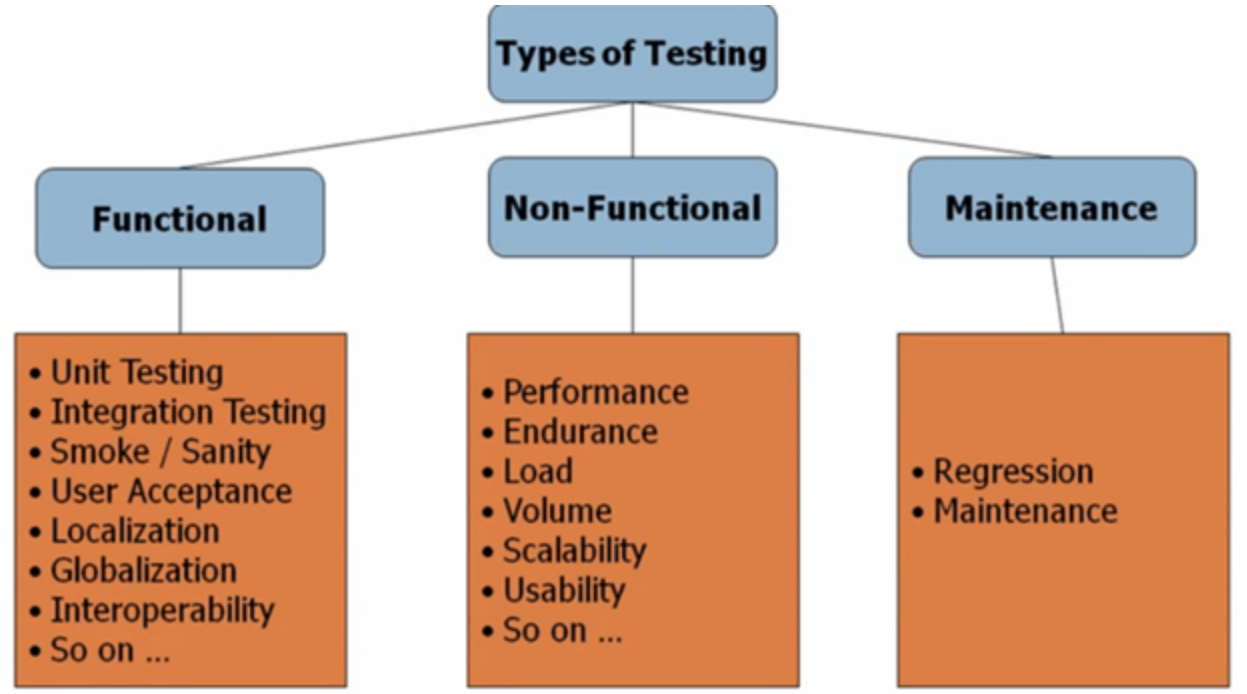


Figure 6.3:Agile QA process

Apart from this technique regression testing was performed for ensuring if the feature operates a desired behaviour after the bug fixes are made. A quick sanity testing to check the output is true and matches the desired one. Non-functional testing to facilitate the performance, availability, load etc for our web application as well as Mobile app.



6.2.2 Test Cases

Following are the end to end functionality test cases for the application -

1)Check validations for user registration

Test Case ID:	PK-TC-01	**Test Type:**	Functional
Test Item:	SignUp Validation	**Product Name:**	PriceKing
Test Case Goal:	To test the validations given to the user input fields.		
Step #	**Description**	**Input**	**Expected Result**

1)	Welcome screen		All the text and graphics must be well aligned
2)	Click on Signup		Opens register page
3)	**Checks** Fill out the details	Provide the required first name and last name field input	User must only be able to enter the characters from 'a-z' or 'A-Z'and no numbers '0-9'or special characters should be accepted.
4)	**Checks** Fill out the details	Provide the required email id input	Only email address format is accepted. eg - -----@-----.com
5)	**Checks** Fill out the details	Provide the required phone number	Allows only (000)000-0000 format
6)	**Checks** Fill out the details	Provide the required password input	-Check if the password is 8 to 20 characters. Password with less than 8 character not accepted and should show a

			message -Check confirm password must match the given password if it doesn't then it shows an error message

Test Case ID:	PK-TC-02	**Test Type:**	Functional
Test Item:	SignUp	**Product Name:**	PriceKing
Test Case Goal:	To test SignUp Functionality		
Steps	**Description**	**Input**	**Expected Result**
1)	Welcome screen		All the text and graphics must be well aligned
2)	Signup	all the required fields	If the user already exists popups a message otherwise redirects to the next page

Test Case ID:	PK-TC-03	**Test Type:**	Functional
Test Item:	SignIn	**Product Name:**	PriceKing
Test Case Goal:	To test the validations given to the user input fields.		
Steps	**Description**	**Input**	**Expected Result**
1)	SignIn screen		Text and graphics are well aligned
2)	**Checks** Fill out the details	user id, password	Searches for a match from database and prompts if the user id/password is incorrect
3)	**Check** signin via facebook,twitter or g+	signin via social media	User should be able to signin via social media like facebook, twitter and linkedin

Test Case ID:	PK-TC-04	**Test Type:**	Functional
Test	Add product	**Product Name:**	PriceKing

Item:	Validation		
Test Case Goal:	To validation on add product modal		
Step #	**Description**	**Input**	**Expected Result**
1)	**Checks** Fill out the details	provide all the required inputs	all the required fields should not be empty
	Checks Fill out the details		url should be in URL format else message is prompted
	Checks Whether product is added to the table	click on save	Product is added to the list of the products along with thumbnail image

Test Case ID:	PK-TC-05	**Test Type:**	Functional
Test Item:	View Product	**Product Name:**	PriceKing
Test Case Goal:	To test the validations on the view product modal		
Steps	**Description**	**Input**	**Expected Result**

1)	**Checks** hover effect on the product name	move the mouse to the product name field	Product name appears in large fonts and underlined
2)	**Checks** Details of the product is displayed in the modal	click on any product name	Modal opens which contains read only details of that particular product
3)	**Checks** Close modal	click on close	the modal closes

Test Case ID:	PK-TC-06	**Test Type:**	Functional
Test Item:	Delete Product	**Product Name:**	PriceKing
Test Case Goal:	To test delete product functionality		
Steps	**Description**	**Input**	**Expected Result**
1)	**Checks** delete a product	click on delete	deletes the product from the database as well as frontend

Test	PK-TC-07	**Test Type:**	Functional

Case ID:			
Test Item:	Edit Product	**Product Name:**	PriceKing
Test Case Goal:	To test the validations on Edit product modal.		
Steps	**Description**	**Input**	**Expected Result**
1)	**Checks** fields should contain the product details	click on edit	fields should not be empty. It should contain details of that particular product
2)	**Checks** Update	click on update	should update the edited information in the table as well as database

Test Case ID:	PK-TC-08	**Test Type:**	Functional
Test Item:	Add coupon Validation	**Product Name:**	PriceKing
Test Case Goal:	To test the validations on the add coupon modal		
Step #	**Description**	**Input**	**Result**

1)	**Checks** Fill out the details	provide all the required inputs	all the required fields should not be empty
	Checks Fill out the details		price should be int/double expiry date should be in the date format
	Checks Whether coupon is added to the table	click on save	Coupon is added to the list of the products along with thumbnail image

Test Case ID:	PK-TC-09	**Test Type:**	Functional
Test Item:	View Coupon	**Product Name:**	PriceKing
Test Case Goal:	To test the view coupon modal.		
Steps	**Description**	**Input**	**Result**
1)	**Checks** hover effect on the coupon name	move the mouse to the coupon name field	coupon name should appear in large fonts and underlined
2)	**Checks** Details of the coupon	click on any coupon name	Modal opens which contains read only details of that

	is displayed in the modal		particular coupon
2)	**Checks** Close modal	click on close	the modal should close

Test Case ID:	PK-TC-10	**Test Type:**	Functional
Test Item:	Delete coupon	**Product Name:**	PriceKing
Test Case Goal:	To test the delete coupon functionality.		
Steps	**Description**	**Input**	**Expected Result**
1)	**Checks** delete a coupon	click on delete	deletes the coupon from the database as well as frontend

Test Case ID:	PK-TC-11	**Test Type:**	Functional
Test Item:	Edit coupon	**Product Name:**	PriceKing
Test	To test the validations and functionality of edit coupon		

Case Goal:			
Steps	**Description**	**Input**	**Expected Result**
1)	**Checks** fields should contain the coupon details	click on edit	fields should not be empty. It should contain details of that particular coupon
2)	**Checks** Update	click on update	should update the edited information in the table as well as database

Test Case ID:	PK-TC-12	Test Type:	Functional
Test Item:	Analytics	Product Name:	PriceKing
Test Case Goal:	To test the analytics charts.		
Steps	**Description**	**Input**	**Expected Result**
1)	**Check** Redirection	click on analytics	It should redirect to the analytics page
2)	**Check** Graphs		Graphs should be displayed properly

Mobile Testing

Projcet Name	PriceKing	Projcet Team:

Mobile Devices	Android Mobile Phone and Tablet	Team Mavericks
OS	Android	
Scope/Module	Search cheap products	

Test Case Id	1
Test Case Name	Splash Screen should be displayed while loading recommendation service in the background and display home screen with recommendations.
Pre Requisite	App should be installed into device.
Test Step	1. Start app by pressing icon 2. Wait for result to be downloaded and see the Home Screen
Actual Result	The app would execute recommendation service in the background while it's showing splash screen in the front. When its downloaded, user can see these recommendations on the Home screen in a gallery.

Test Case Id	2
Test Case Name	Provide search query using one of the input ways viz. text, voice and image and the app would display a list of cheap products.
Pre Requisite	1. App should be installed into device. 2. It should have internet connectivity.
Test Step	1. Start app by pressing icon 2. Provide input query on home screen and wait for the result.
Actual Result	The app would pick the input parameter provided by text, voice or image and hit the web service request. It would get the response, parse it and finally display in the list view.

Test Case Id	3
Test Case Name	Share deals on social networking sites.
Pre Requisite	1. App should be installed into device. 2. It should have internet connectivity. 3. The device must have particular social networking app installed.
Test Step	1. Start app by pressing icon 2. Go to Product detail screen and share it with Facebook, twitter, LinkedIn or email.
Actual Result	The app successfully shares the product details on selected social networking platform provided the app is installed.

Test Case Id	4
Test Case Name	Add Reminder for a Product.
Pre Requisite	1. App should be installed into device.
Test Step	1. Start app by pressing icon 2. Go to Product detail screen and hit “Add Reminder” button and the app would add that product in Calendar.
Actual Result	The app successfully adds a reminder with the selected date and time in the device’s calendar.

Test Case Id	5
Test Case Name	Buy Product
Pre Requisite	1. App should be installed into device.
Test Step	1. Start app by pressing icon 2. Go to Product detail screen and hit “Buy” button and the app would take you to the Product on the device’s browser with the help of product url.

Actual Result	The app takes you to the actual web site in your device's browser so that user can buy the product online.

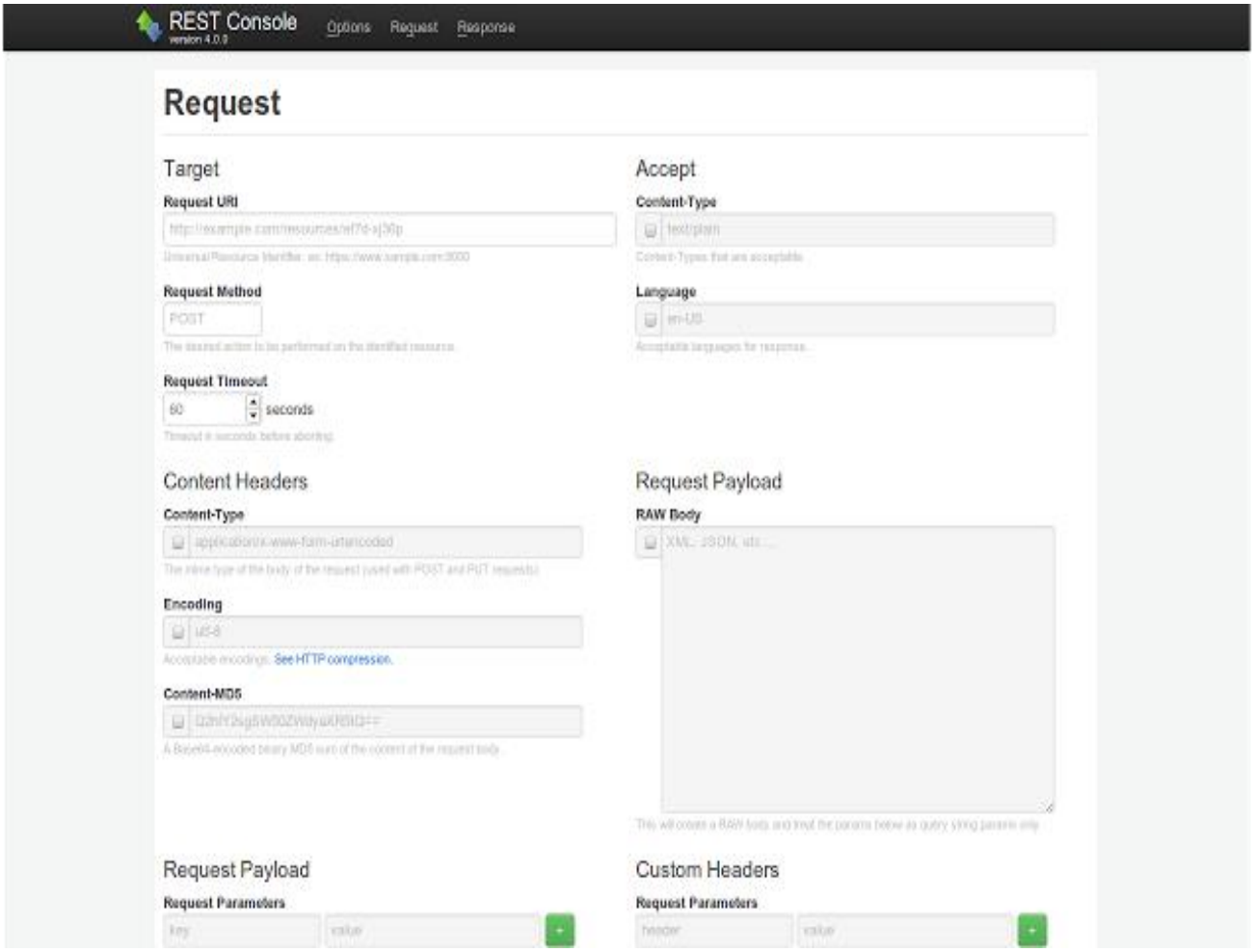
Test Case Id	6
Test Case Name	Add to Wish List
Pre Requisite	1. App should be installed into device. 2. The user must be signed in
Test Step	1. Start app by pressing icon 2. Sign in first. 3. Search any product of your choice and go to detail screen. 4. Press "Add to Wish List" button.
Actual Result	Open the menu and go to wish list activity and you will see all the items added to wish list provided you are logged in.

Test Case Id	7
Test Case Name	Recently Viewed Items
Pre Requisite	1. App should be installed into device.
Test Step	1. Start app by pressing icon 2. View any item by going to product detail screen.
Actual Result	Open the app and go to any product detail screen and select the "Recently Viewed Items" in the category menu option. You will see all the recently viewed items in the app.

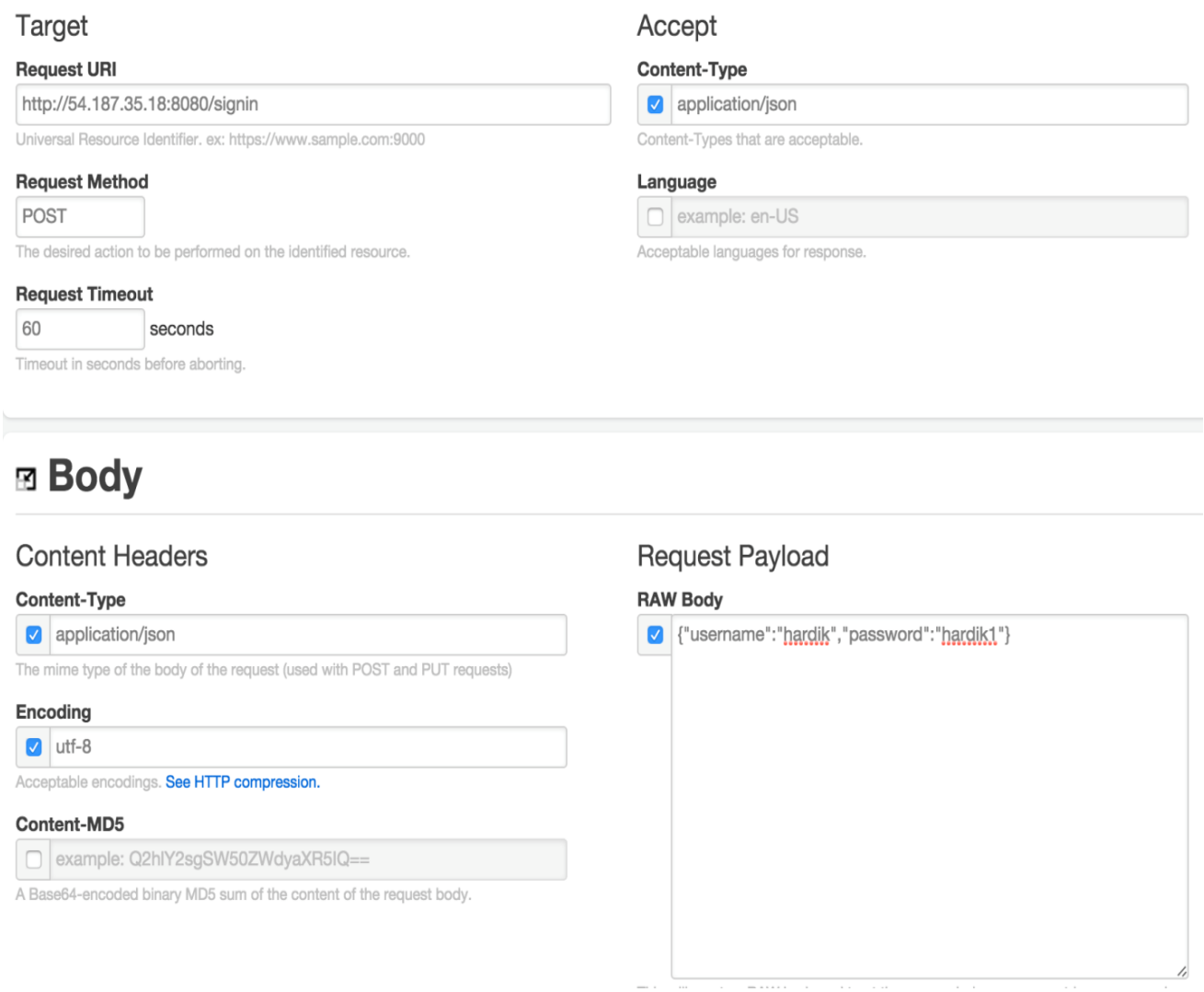
Test Case Id	8
Test Case Name	Sign In and Sign Up
Pre Requisite	1. App should be installed into device. 2. Internet connectivity is required.

Test Step	1. Start app by pressing icon 2. Go to “Sign In” option in the category menu option. For the first time, select “Create an Account” button and register yourself with the app.
Actual Result	The validation would be done while Sign Up and Sign In process and user would be authenticated.

6.2.3 Middle Tier and UI testing using Rest Console

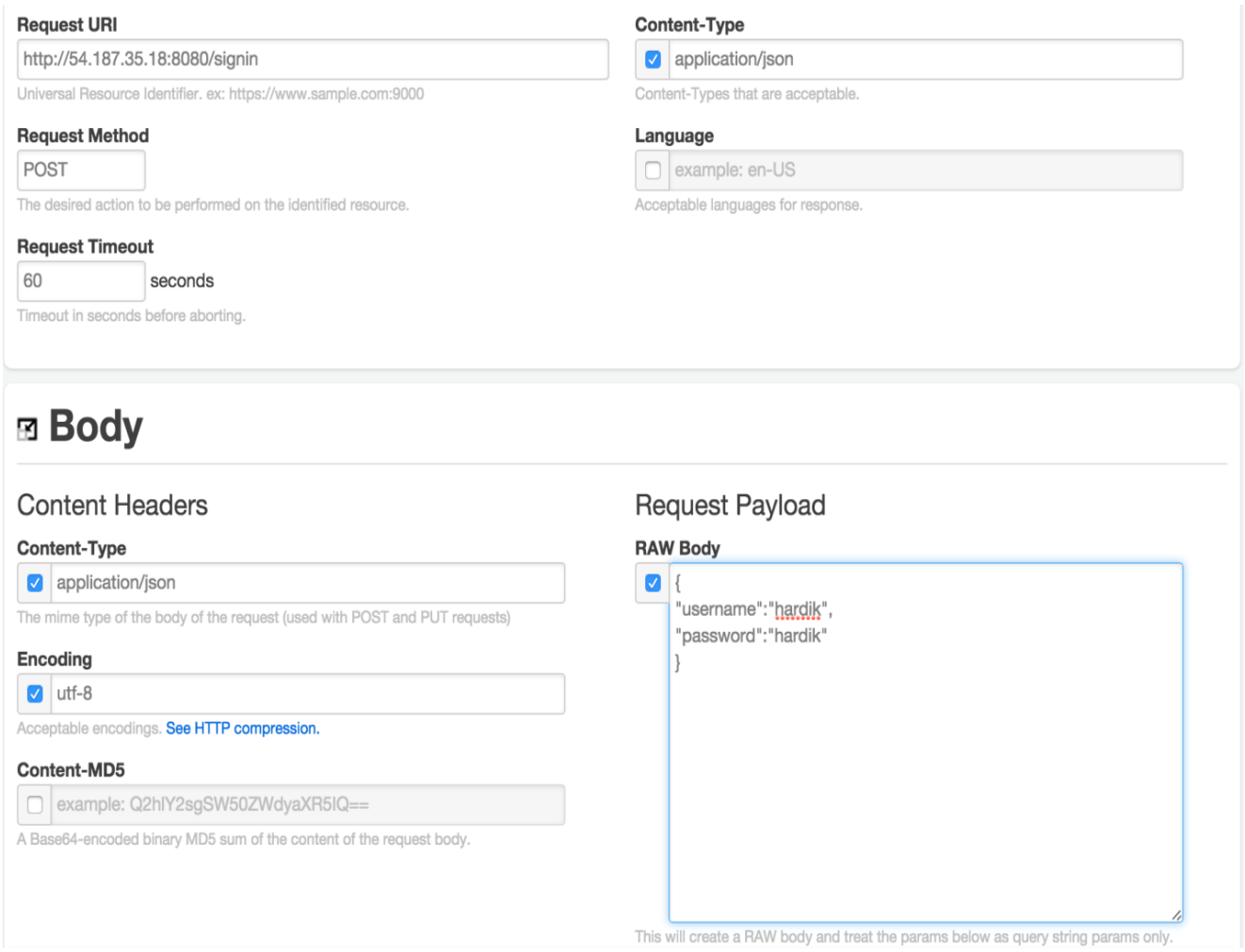


SignIn incorrect password



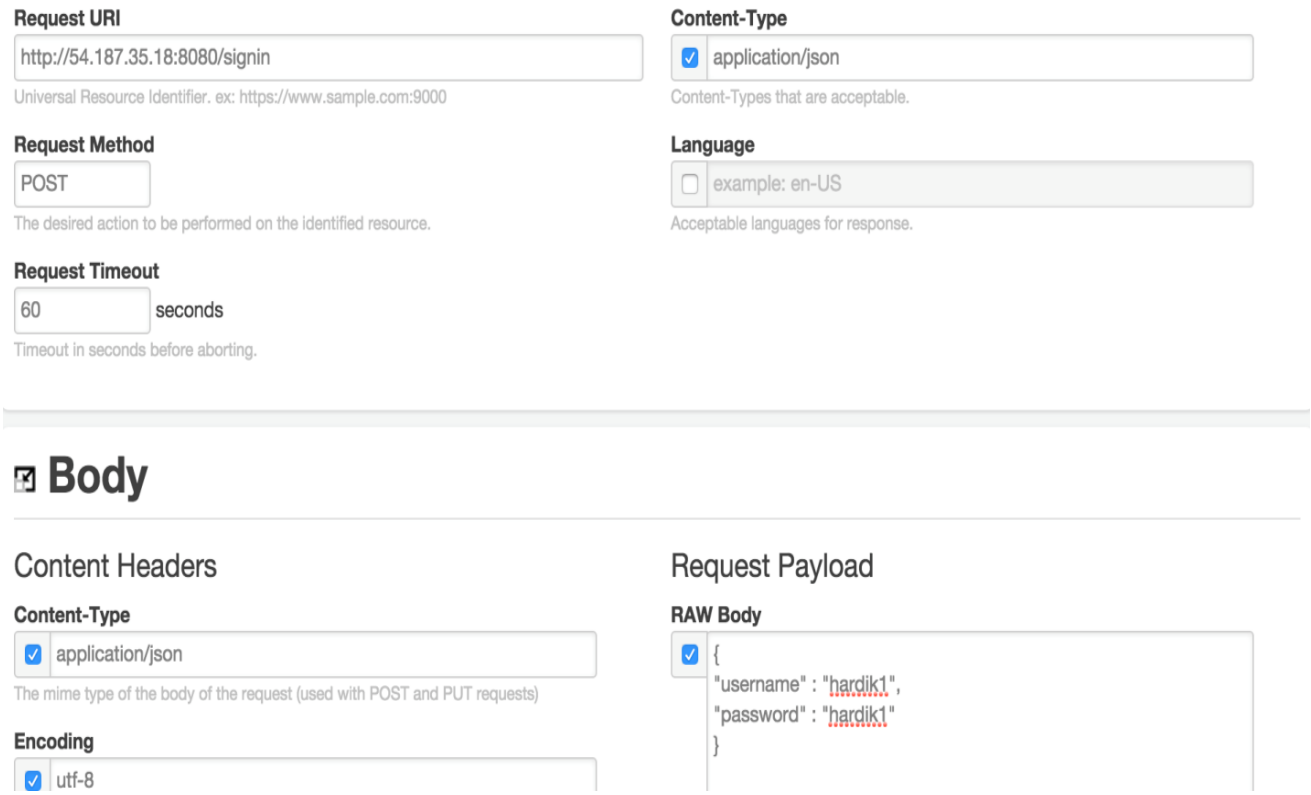

```
{
    "errCode": 1,
    "errorMessage": "Incorrect Password",
    "uid": "hardik"
}
```


Sign In correct password



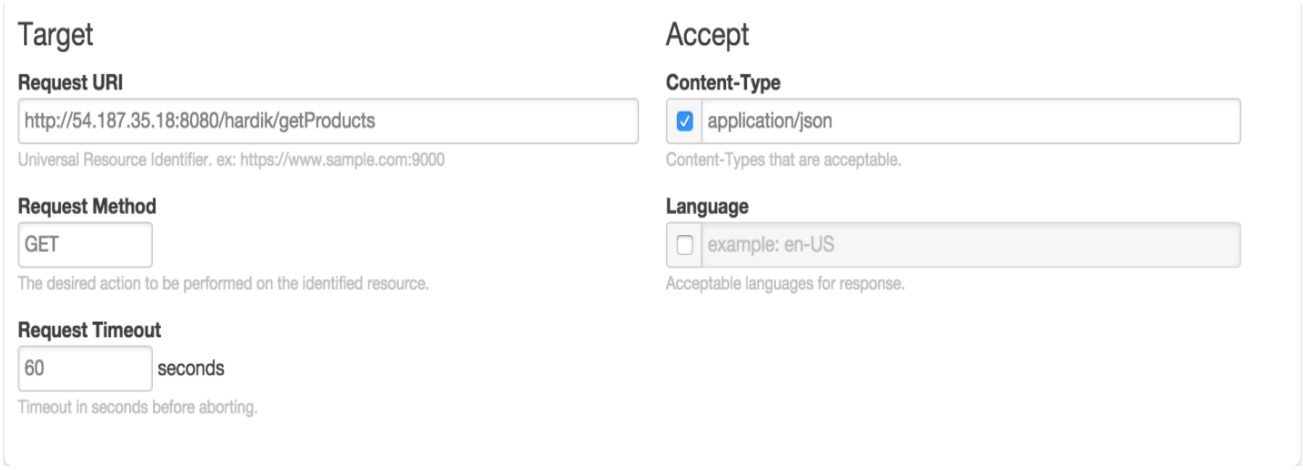

```
{
    "errCode": 0,
    "errorMessage": "success",
    "uid": "hardik",
    "email": "hardikjoshi.se@gmail.com"
}
```


Sign in incorrect username



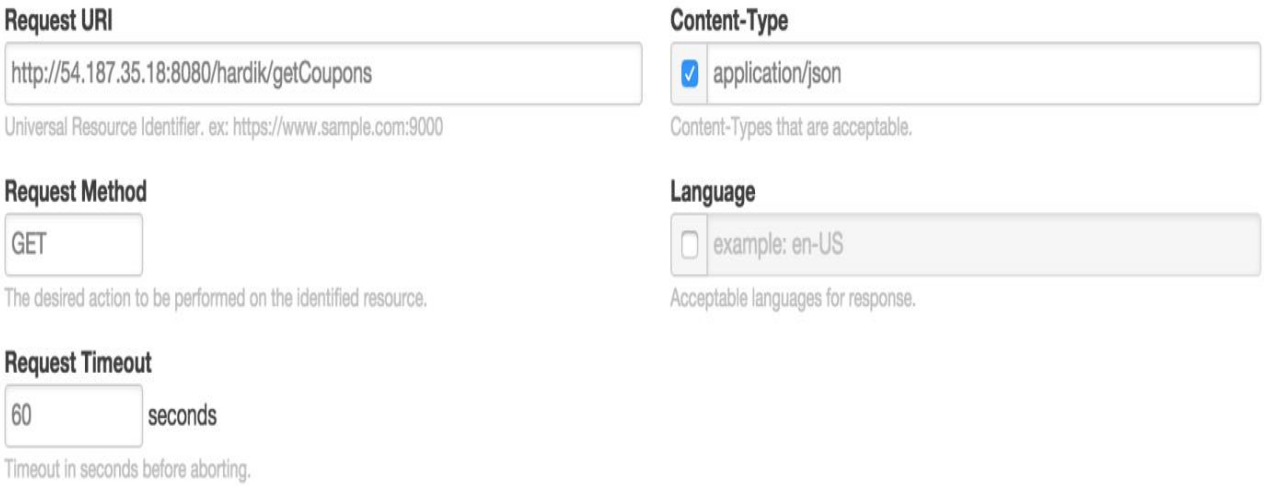

```
1. {
2.     "errCode": 1,
3.     "errorMessage": "Incorrect Username",
4.     "uid": "hardik1"
5. }
```

Get products



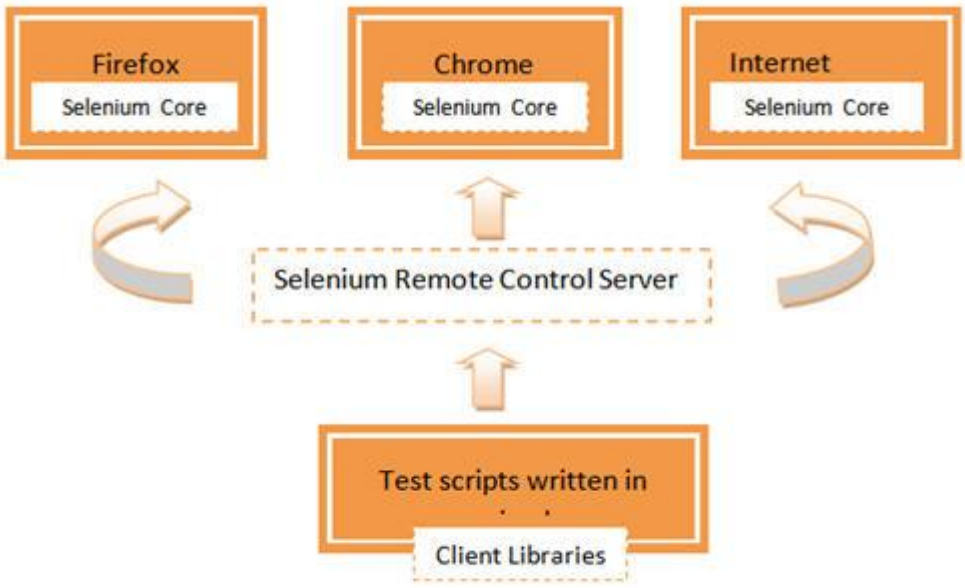

```
[{
    "_id": 2,
    "productName": "iPhone6",
    "productDescription": "iPhone6 64 GB Silver",
    "productCategory": "Electronics",
    "thumbnailImage": "",
    "productUrl": "N/A",
    "price": "699",
    "uploadedBy": "hardik",
    "isDeleted": false
}, {
    "_id": 3,
    "productName": "Bed Frame",
    "productDescription": "This is best price in the market.",
    "productCategory": "Furniture",
    "thumbnailImage": "",
    "productUrl": "N/A",
    "price": "40",
    "uploadedBy": "hardik",
    "isDeleted": false
}]
```


Get coupons

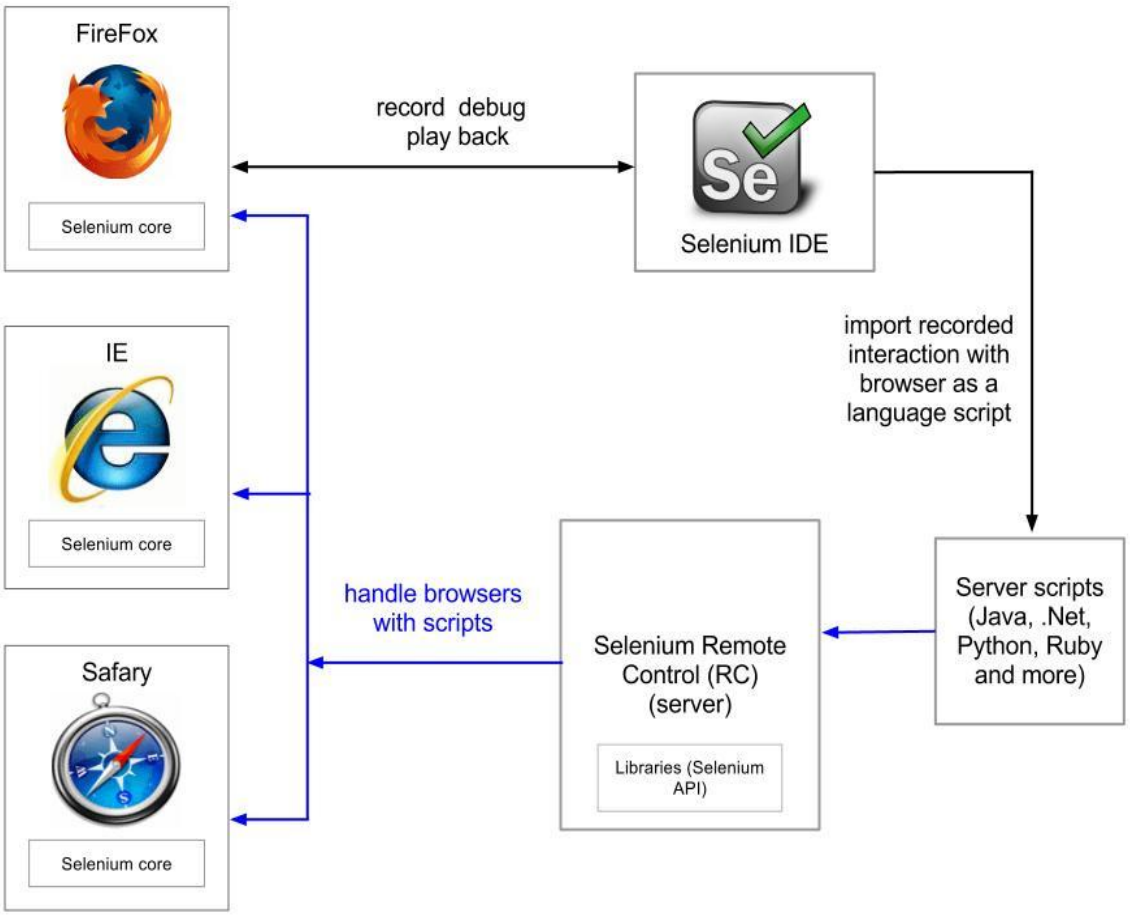



```
[{
    "_id": 2,
    "name": "iPhone 6 Case",
    "description": "This offer is for limited time. Hurry Up! - 50% off on eBay",
    "originalPrice": 20.0,
    "discountedPrice": 10.0,
    "couponCode": "HGY87RIO",
    "expiryDate": "May 9, 2015 7:00:00 AM",
    "uploadedBy": "hardik",
    "isDeleted": false
}]
```

6.2.4 Functionality testing using Selenium IDE



Selenium RC Supports - © www.SoftwareTestingHelp.com


FireFox
Selenium core
record debug
play back
Selenium IDE
import recorded
interaction with
browser as a
language script
IE
Selenium core
handle browsers
with scripts
Selenium Remote
Control (RC)
(server)
Libraries (Selenium
API)
Server scripts
(Java, .Net,
Python, Ruby
and more)
Safary
Selenium core

```java
package com.priceking.cmpe295B;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class test {
    public static void main(String... args) throws InterruptedException{



        System.setProperty("webdriver.chrome.driver", "//usr//local//share//chromedriver");
           WebDriver driver = new ChromeDriver();
       driver.get("http://localhost:8080/signin");
       WebElement username = driver.findElement(By.id("uname"));
       username.sendKeys("nshah");
       Thread.sleep(3000);
       WebElement password = driver.findElement(By.id("pass"));
       password.sendKeys("naiya");
       Thread.sleep(3000);
       WebElement submit = driver.findElement(By.id("submitid"));
       submit.click();
       Thread.sleep(3000);

       WebElement testelement = driver.findElement(By.xpath("//*[@id='page-wrapper']/div[1]/div/h1"));
       Thread.sleep(6000);
       if(testelement.getText().equalsIgnoreCase("Welcome!"))
       {
           System.out.println("Test successful");

       }
       //add product
```

```java
//add product
WebElement addproduct = driver.findElement(By.id("addproduct"));
addproduct.click();
Thread.sleep(3000);
WebElement pname = driver.findElement(By.id("pname"));
pname.sendKeys("iphone");
Thread.sleep(3000);
WebElement price = driver.findElement(By.id("pprice"));
price.sendKeys("700");
Thread.sleep(3000);
WebElement url = driver.findElement(By.id("purl"));
url.sendKeys("www.kkkkkjjjj.com");
Thread.sleep(3000);
WebElement category = driver.findElement(By.id("pcategory"));
category.sendKeys("Electronics");
Thread.sleep(3000);
WebElement description = driver.findElement(By.id("pdescription"));
description.sendKeys("iphone");
Thread.sleep(3000);
WebElement image = driver.findElement(By.id("pimage"));
image.sendKeys("/home/nshah/Downloads/abc.jpg");
Thread.sleep(3000);
WebElement submitproduct = driver.findElement(By.id("psave"));
submitproduct.click();
Thread.sleep(3000);
```

```
    Thread.sleep(3000);
    //add coupon
    WebElement addcoupon = driver.findElement(By.id("addcoupon"));
    addcoupon.click();
    Thread.sleep(3000);
    WebElement cname = driver.findElement(By.id("cname"));
    cname.sendKeys("iphone20%");
    Thread.sleep(3000);
    WebElement cdescription = driver.findElement(By.id("cdescription"));
    cdescription.sendKeys("iphone20%");
    Thread.sleep(3000);
    WebElement coriginalprice = driver.findElement(By.id("coriginalprice"));
    coriginalprice.sendKeys("700");
    Thread.sleep(3000);
    WebElement cdiscountedprice = driver.findElement(By.id("cdiscountedprice"));
    cdiscountedprice.sendKeys("600");
    Thread.sleep(3000);
    WebElement ccode = driver.findElement(By.id("ccode"));
    ccode.sendKeys("ABC");
    Thread.sleep(3000);
    WebElement cdate = driver.findElement(By.id("cdate"));
    cdate.sendKeys("25-April-2015");
    Thread.sleep(3000);
    WebElement csave = driver.findElement(By.id("csave"));
    csave.click();
    Thread.sleep(3000);
}
```

```
package com.priceking.cmpe295B;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class test1 {
    public static void main(String... args) throws InterruptedException{

         System.setProperty("webdriver.chrome.driver", "//usr//local//share//chromedriver");
            WebDriver driver = new ChromeDriver();
        driver.get("http://localhost:8080/nshah/home");
        WebElement editproduct = driver.findElement(By.id("peditbutton"));
        editproduct.click();
        Thread.sleep(3000);
        WebElement pname = driver.findElement(By.id("pname"));
        pname.sendKeys("edit1");
        Thread.sleep(3000);
        WebElement psave = driver.findElement(By.id("pedit"));
        psave.click();
        Thread.sleep(3000);

        WebElement editcoupon = driver.findElement(By.id("ceditbutton"));
        editcoupon.click();
        Thread.sleep(3000);
        WebElement cname = driver.findElement(By.id("cname"));

        cname.sendKeys("edit1");
        Thread.sleep(3000);
        WebElement csave = driver.findElement(By.id("cedit"));
        csave.click();
        Thread.sleep(3000);
    }
}
```

6.2.5 Mobile App Automation Testing using Testdroid

Robotium is an Android test automation framework that supports both native as well as hybrid applications. Robotium is generally used for black box testing that is very powerful and robust. Robotium supports Android API level 8 and above. Moreover, this automation tool supports on real devices also. Furthermore, Robotium not require source code every time. If developer has apk file, then also this tool supports automation testing. With support of this open source tool, provided by Apache, developer can write functional, system and acceptance test case scenarios that can span among multiple activities within application.

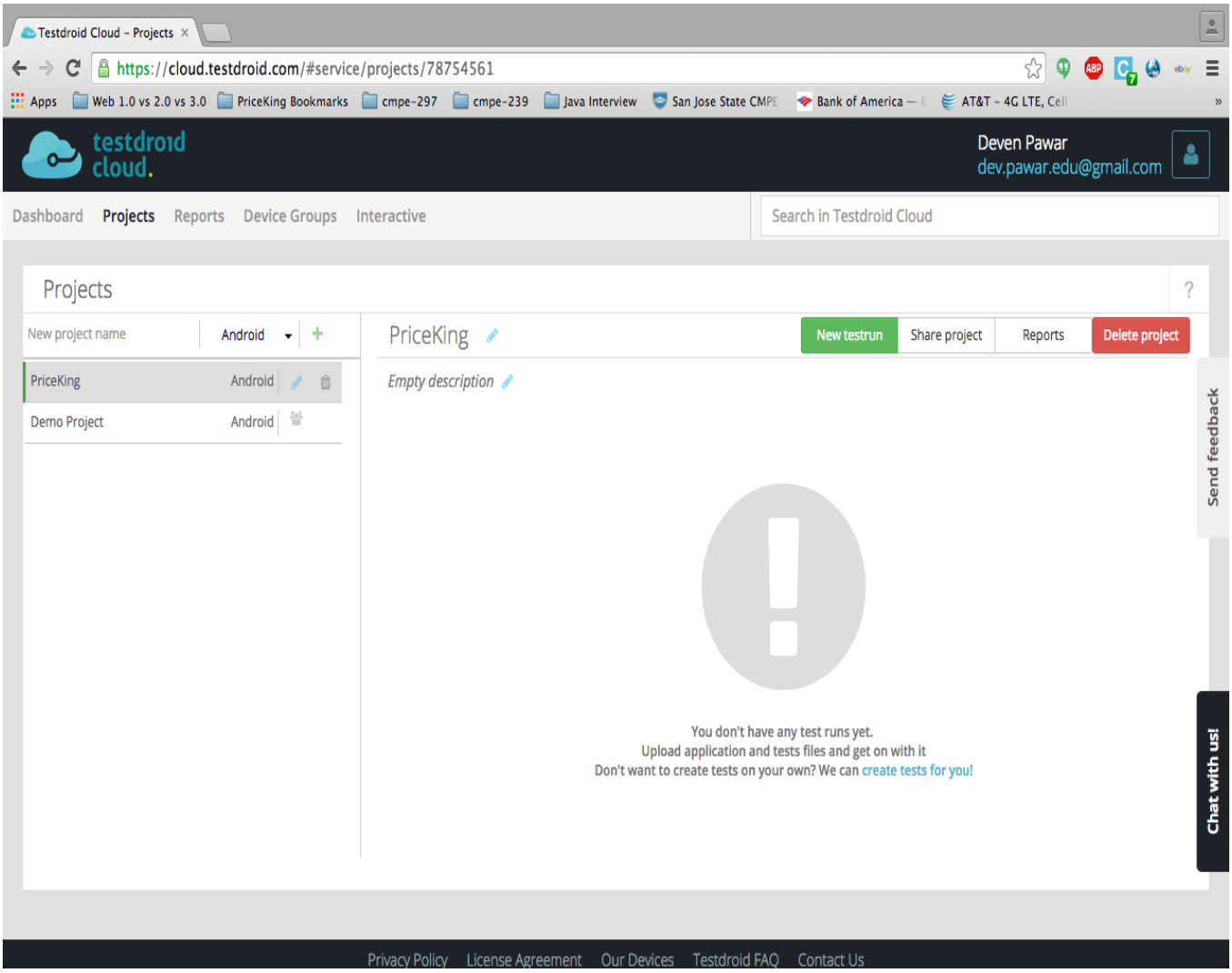
To be very specific, Robotium has following advantages:

1. It supports both native and hybrid apps.
2. This framework is smart enough to handles multiple testing activities automatically.
3. In order to write test cases, this technology requires minimum time.
4. Test cases are more robust compared to other automation testing techniques.

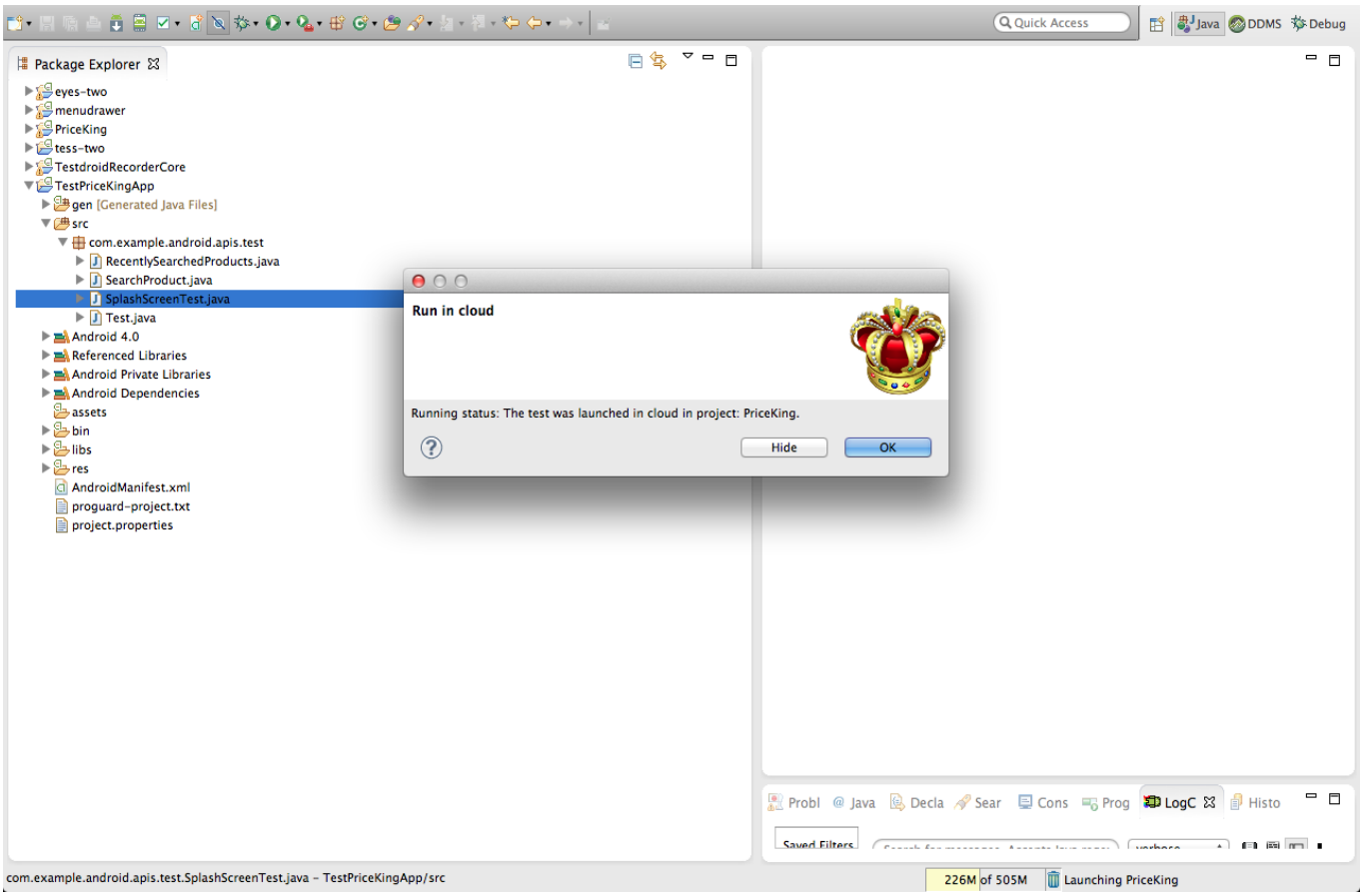
This automation testing can be integrated with Maven, Gradle or Ant to run tests as a part of continuous integration.

We have performed local testing using Android JUnit as well as cloud testing of testdroid cloud. Please find the following screen captures explaining test automation on both local and cloud:

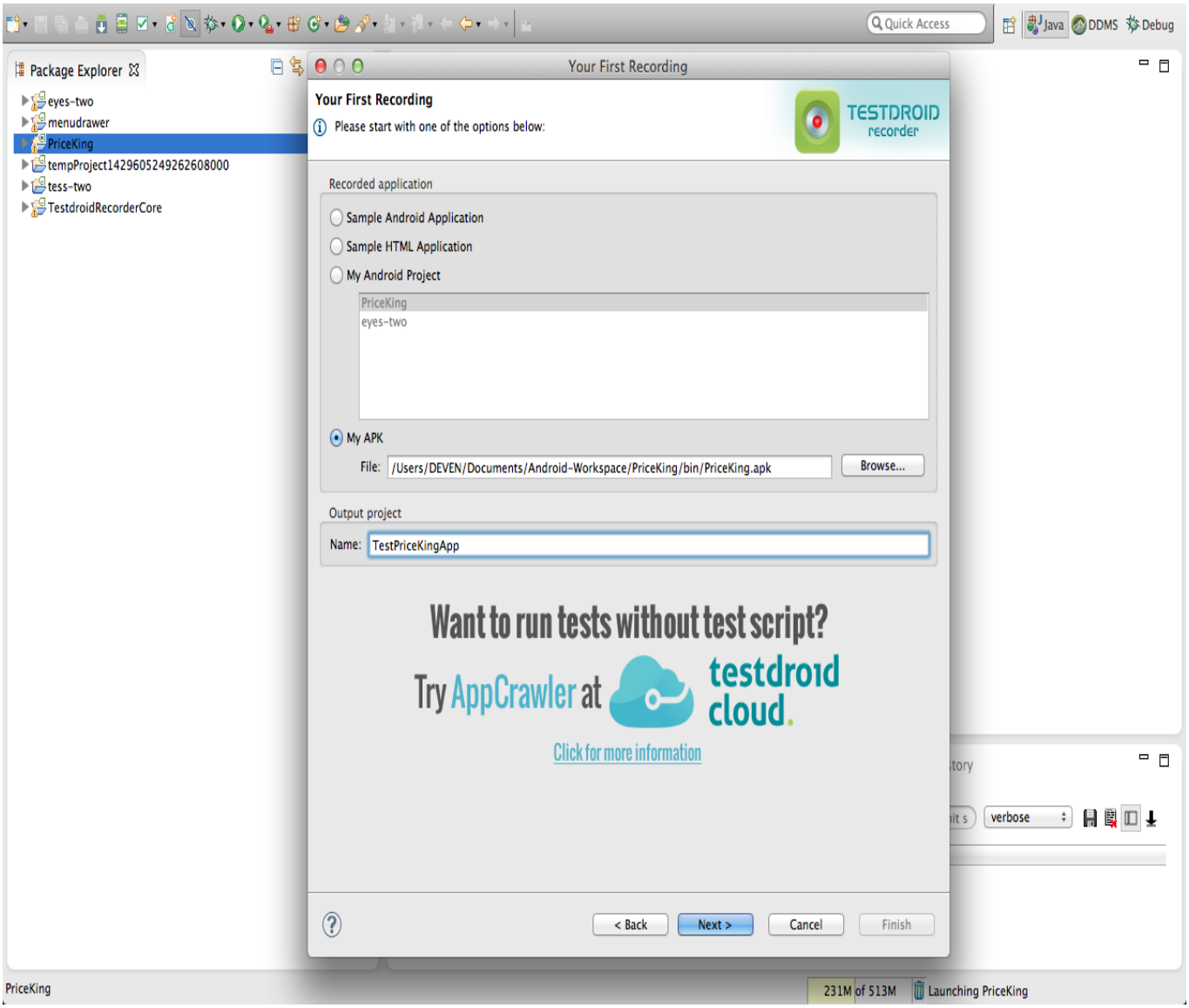
Testing on cloud



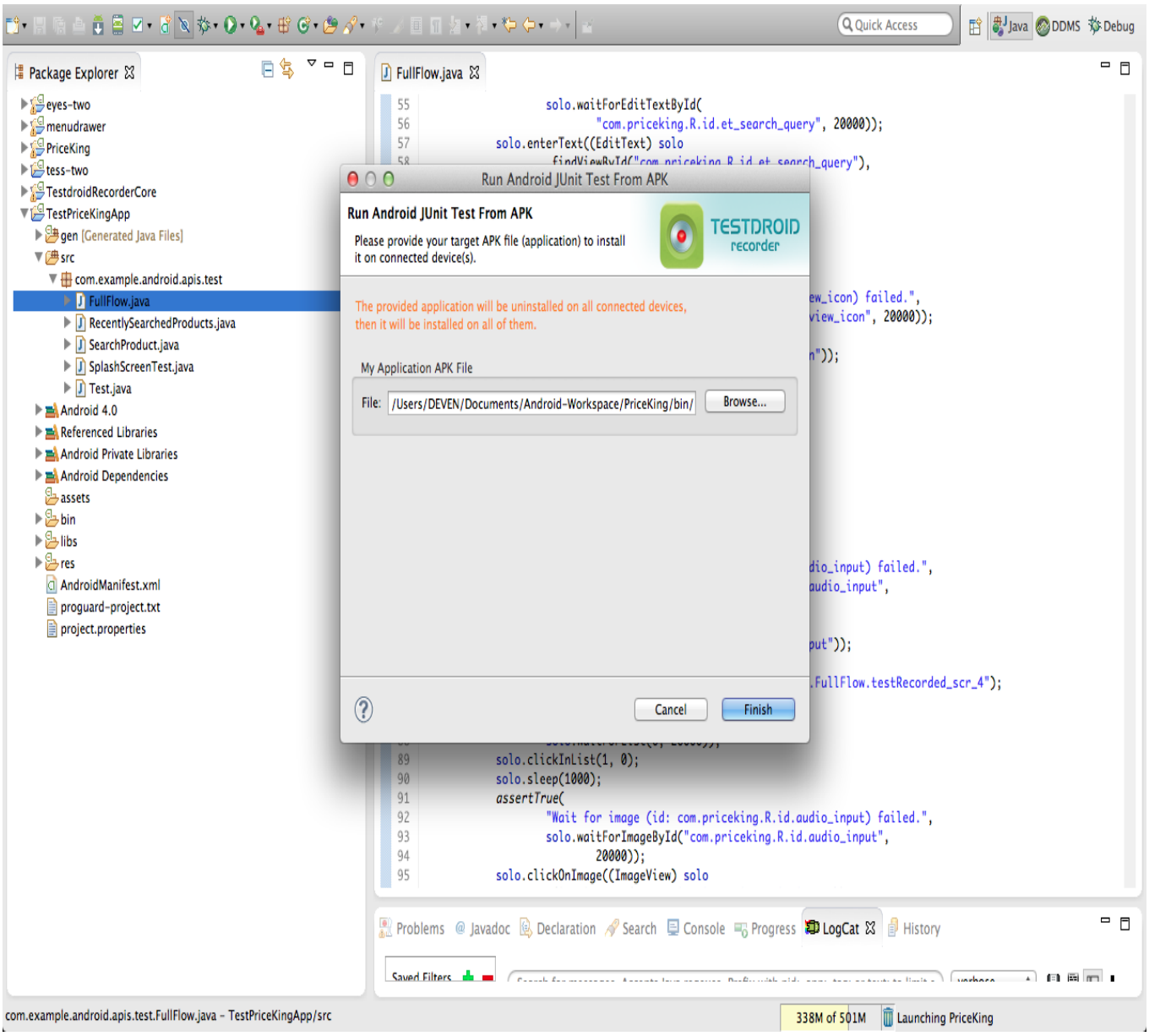
Started Cloud Testing



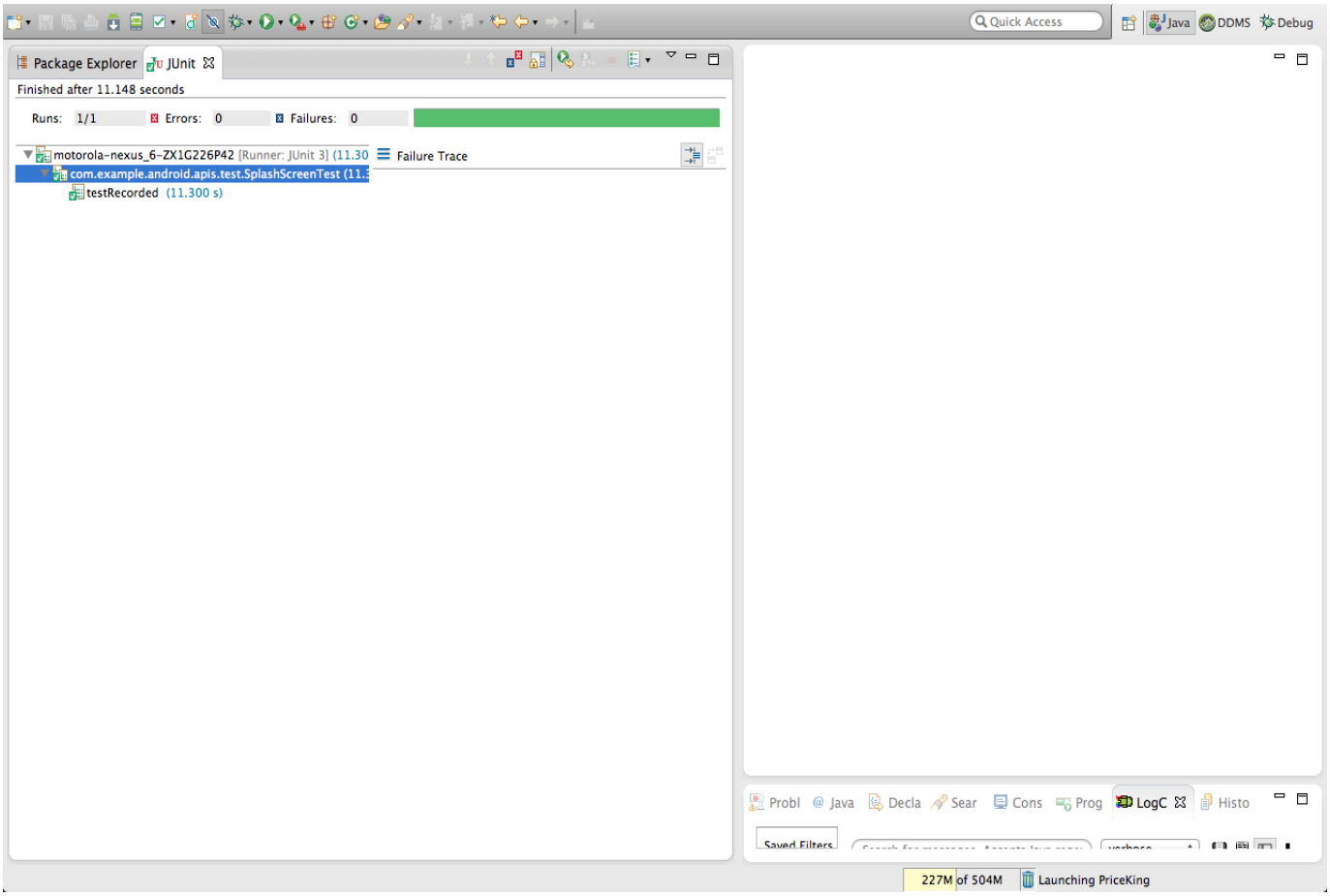
Creating Test Project



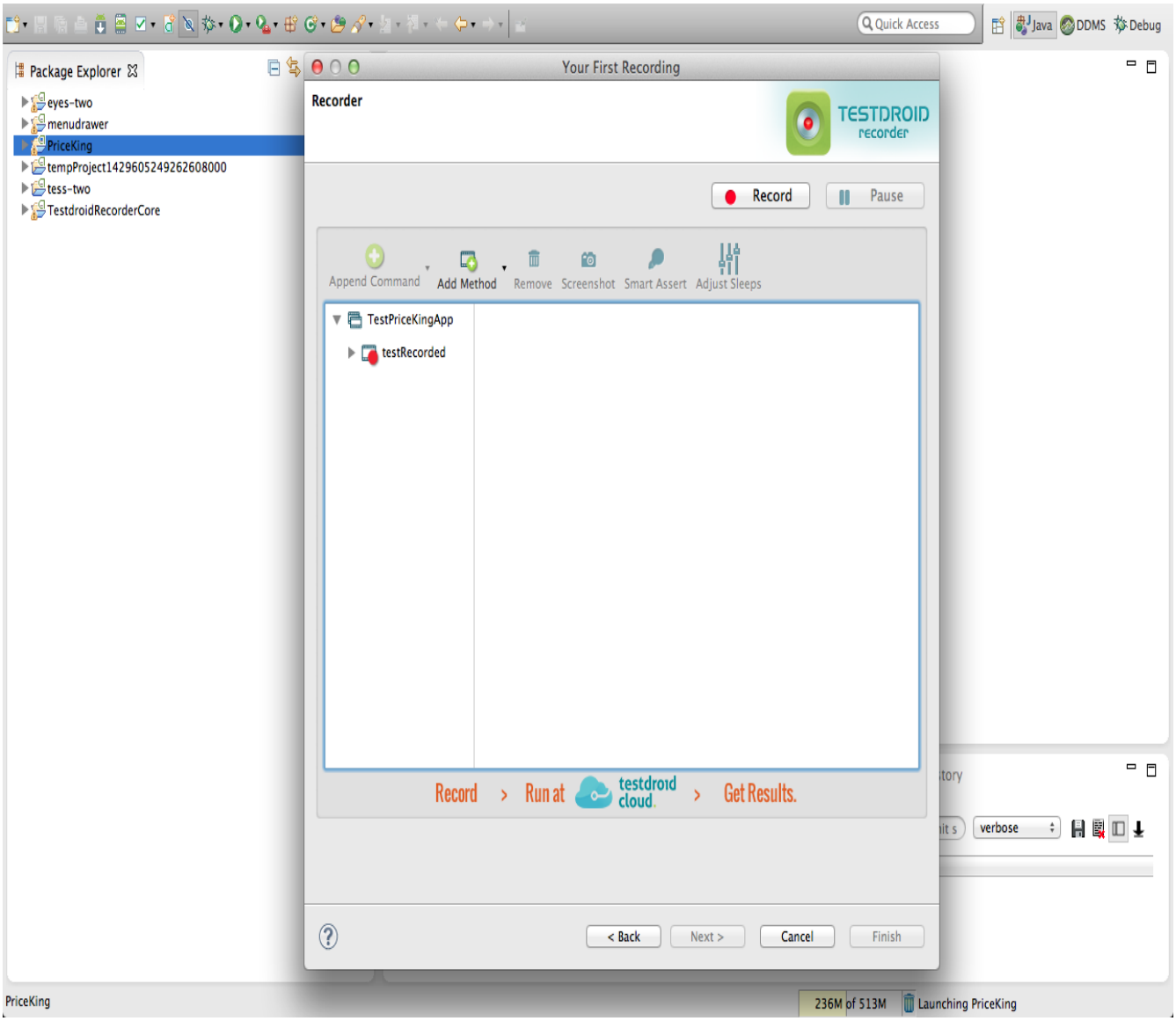
Testing using android Junit



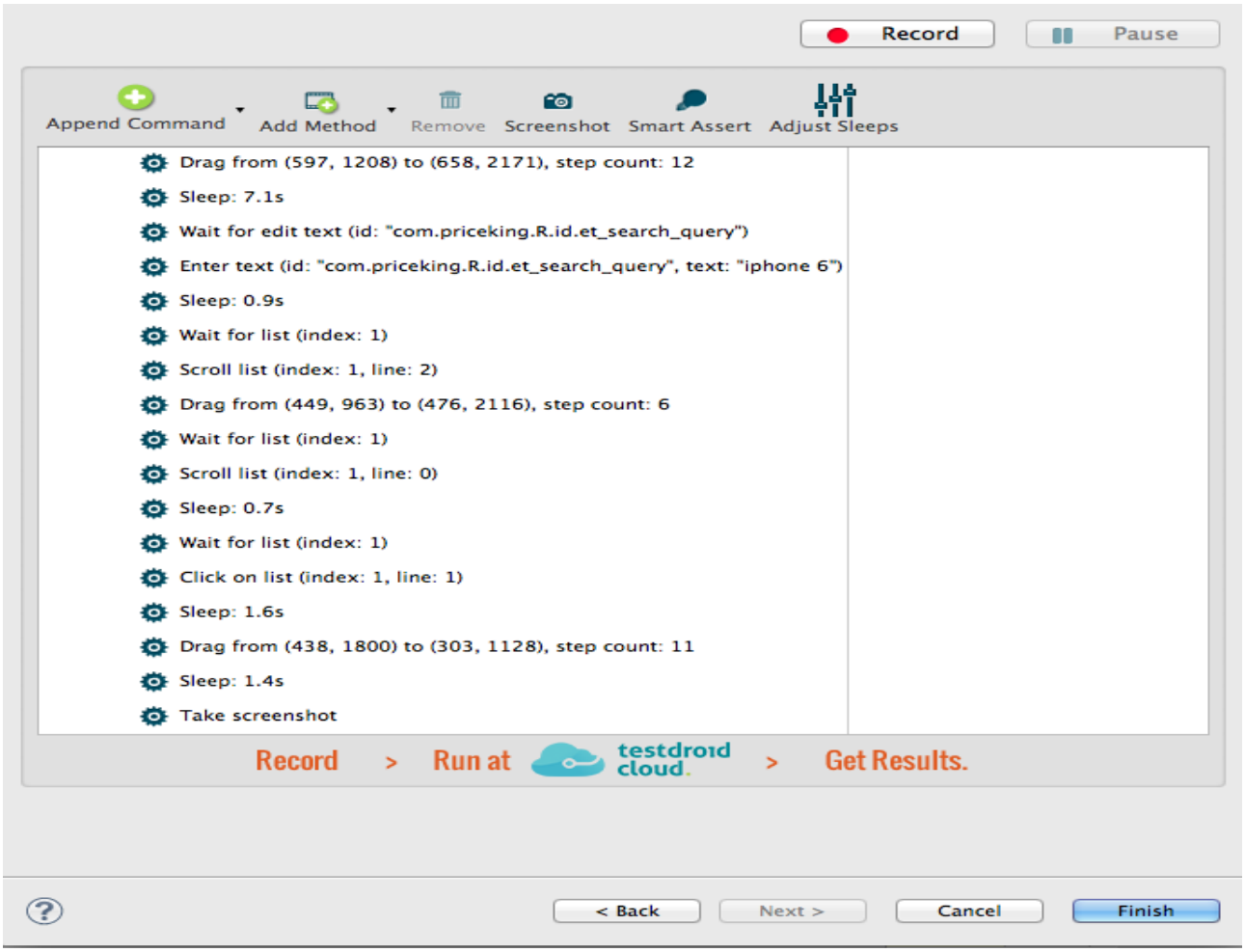
Test case passed android Junit



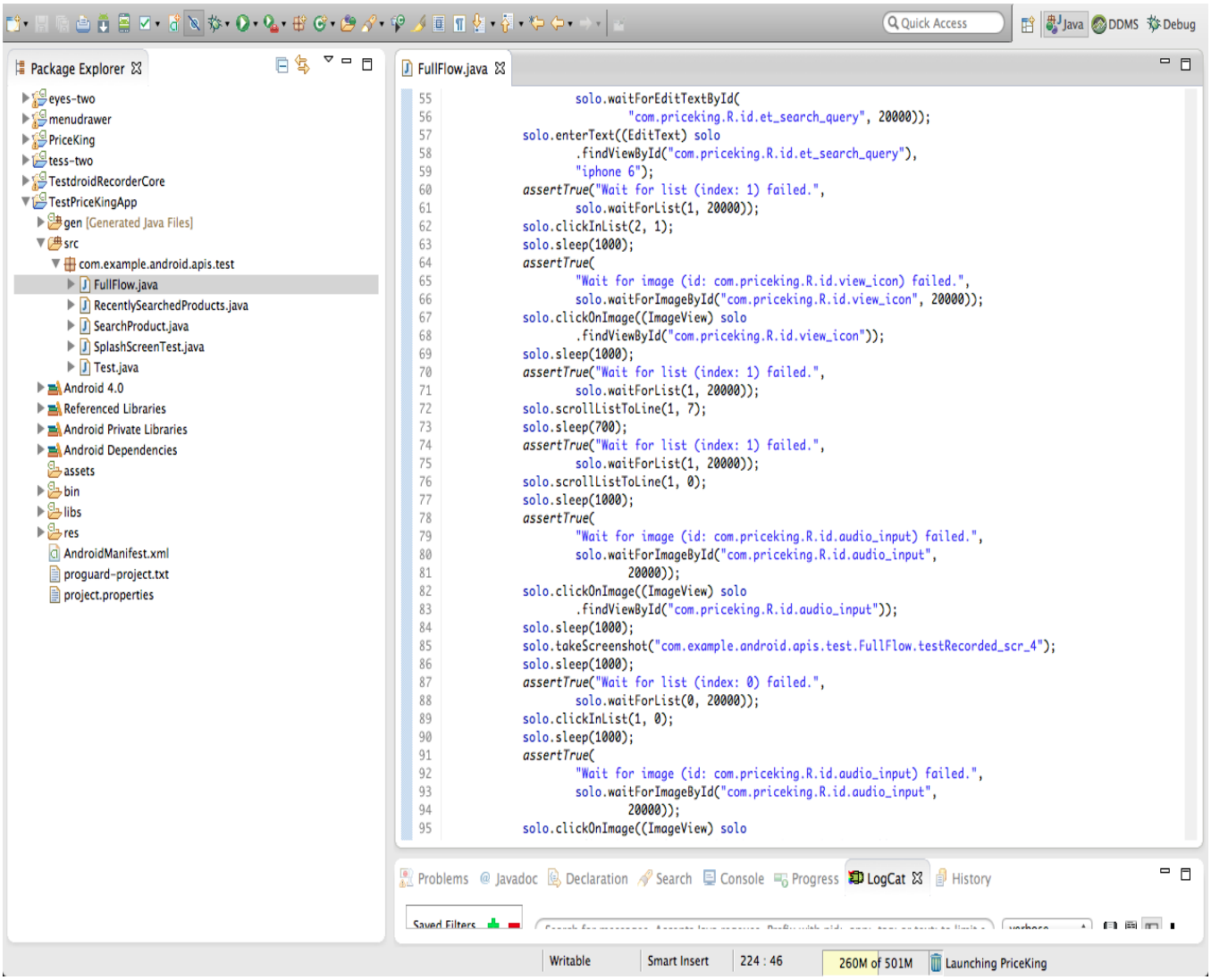
Start Recording



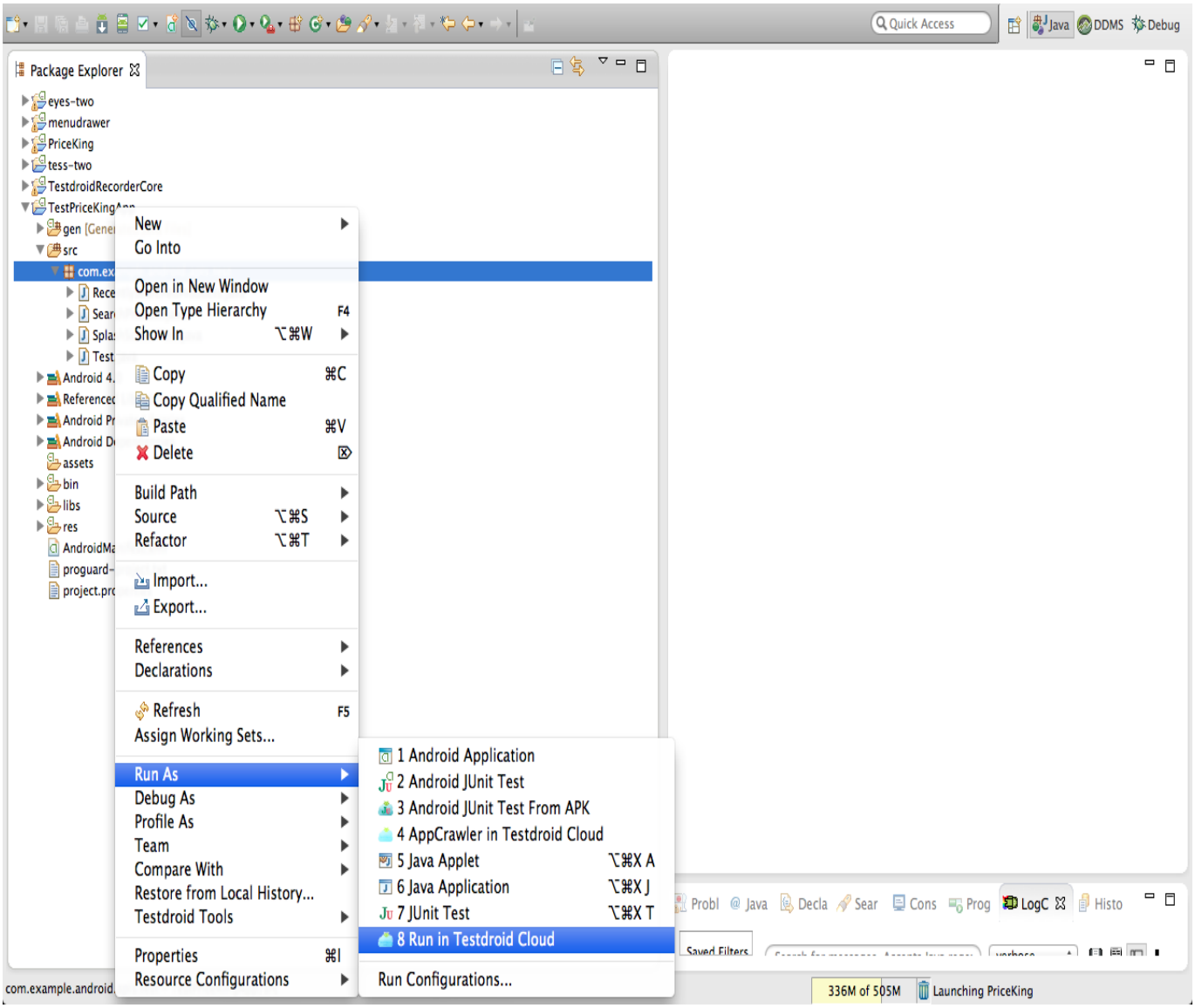
Logs after recording



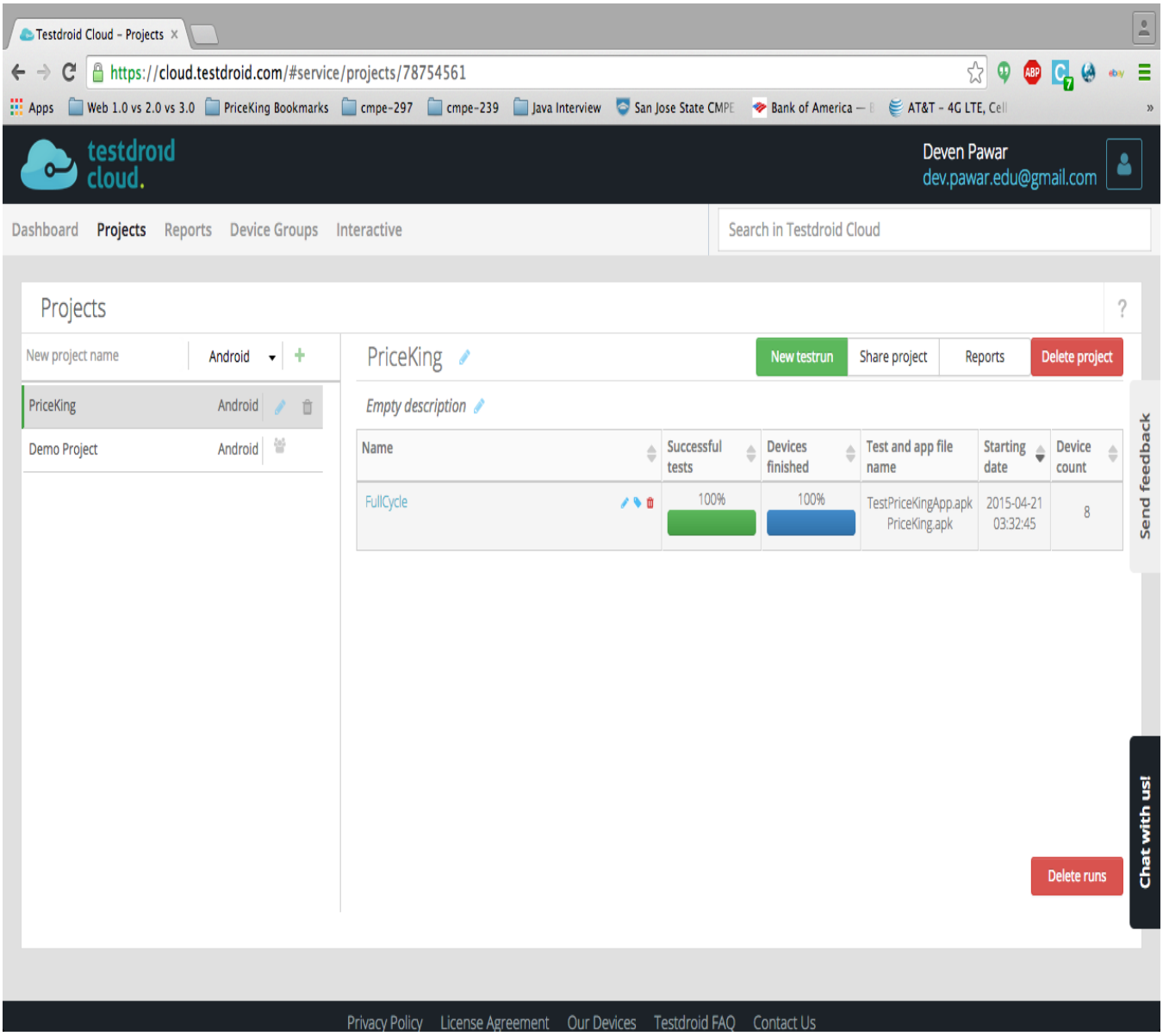
Asset of Full Flow


Package Explorer
eyes-two
menudrawer
PriceKing
tess-two
TestdroidRecorderCore
TestPriceKingApp
gen [Generated Java Files]
src
com.example.android.apis.test
FullFlow.java
RecentlySearchedProducts.java
SearchProduct.java
SplashScreenTest.java
Test.java
Android 4.0
Referenced Libraries
Android Private Libraries
Android Dependencies
assets
bin
libs
res
AndroidManifest.xml
proguard-project.txt
project.properties
FullFlow.java
solo.waitForEditTextById(
"com.priceking.R.id.et_search_query", 20000));
solo.enterText((EditText) solo
.findViewById("com.priceking.R.id.et_search_query"),
"iphone 6");
assertTrue("Wait for list (index: 1) failed.",
solo.waitForList(1, 20000));
solo.clickInList(2, 1);
solo.sleep(1000);
assertTrue(
"Wait for image (id: com.priceking.R.id.view_icon) failed.",
solo.waitForImageById("com.priceking.R.id.view_icon", 20000));
solo.clickOnImage((ImageView) solo
.findViewById("com.priceking.R.id.view_icon"));
solo.sleep(1000);
assertTrue("Wait for list (index: 1) failed.",
solo.waitForList(1, 20000));
solo.scrollListToLine(1, 7);
solo.sleep(700);
assertTrue("Wait for list (index: 1) failed.",
solo.waitForList(1, 20000));
solo.scrollListToLine(1, 0);
solo.sleep(1000);
assertTrue(
"Wait for image (id: com.priceking.R.id.audio_input) failed.",
solo.waitForImageById("com.priceking.R.id.audio_input",
20000));
solo.clickOnImage((ImageView) solo
.findViewById("com.priceking.R.id.audio_input"));
solo.sleep(1000);
solo.takeScreenshot("com.example.android.apis.test.FullFlow.testRecorded_scr_4");
solo.sleep(1000);
assertTrue("Wait for list (index: 0) failed.",
solo.waitForList(0, 20000));
solo.clickInList(1, 0);
solo.sleep(1000);
assertTrue(
"Wait for image (id: com.priceking.R.id.audio_input) failed.",
solo.waitForImageById("com.priceking.R.id.audio_input",
20000));
solo.clickOnImage((ImageView) solo
Problems
Javadoc
Declaration
Search
Console
Progress
LogCat
History
Saved Filters
Writable
Smart Insert
224 : 46
260M of 501M
Launching PriceKing

Run as Cloud



Cloud Test Passed



Cloud Testing Status

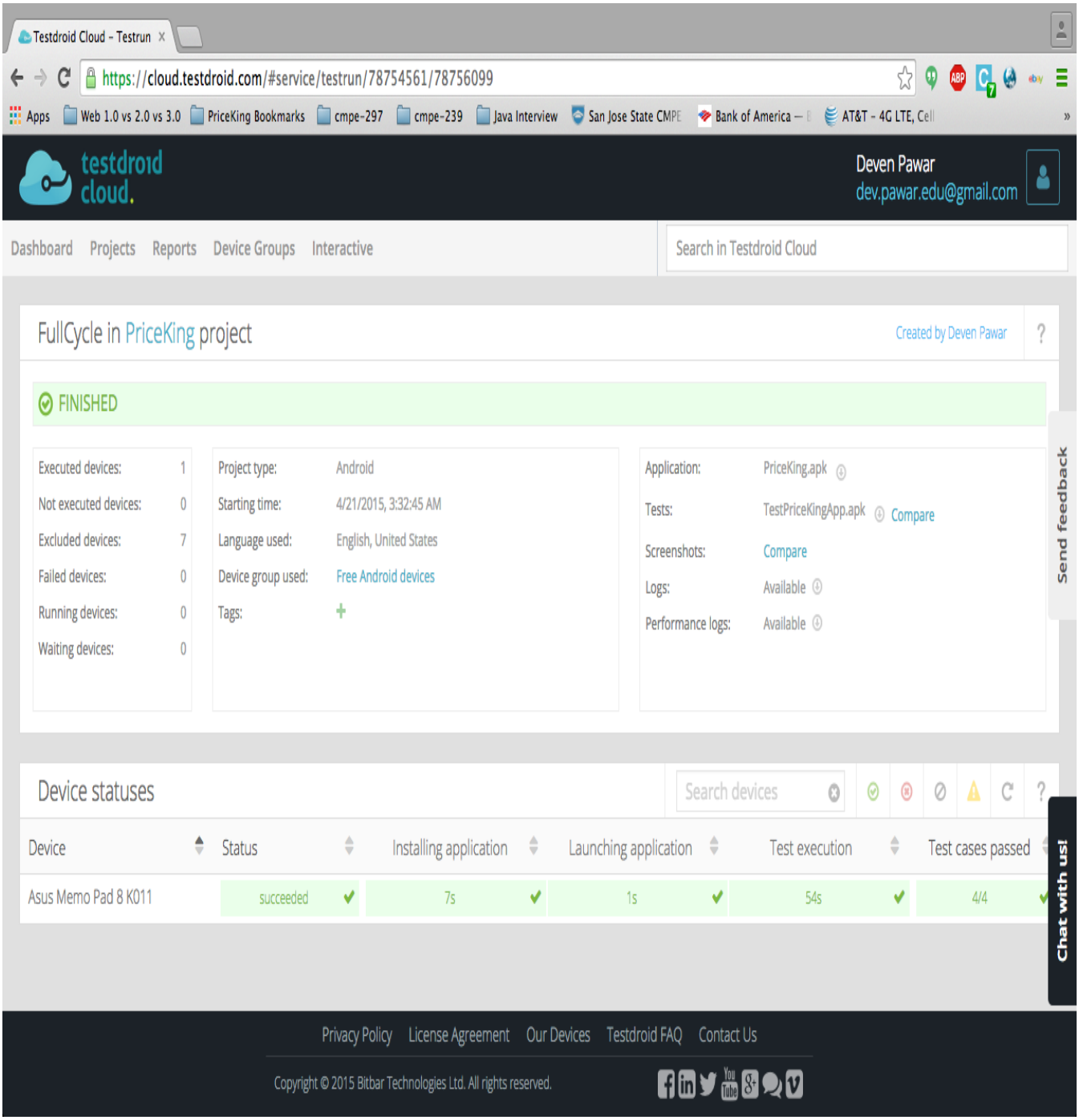
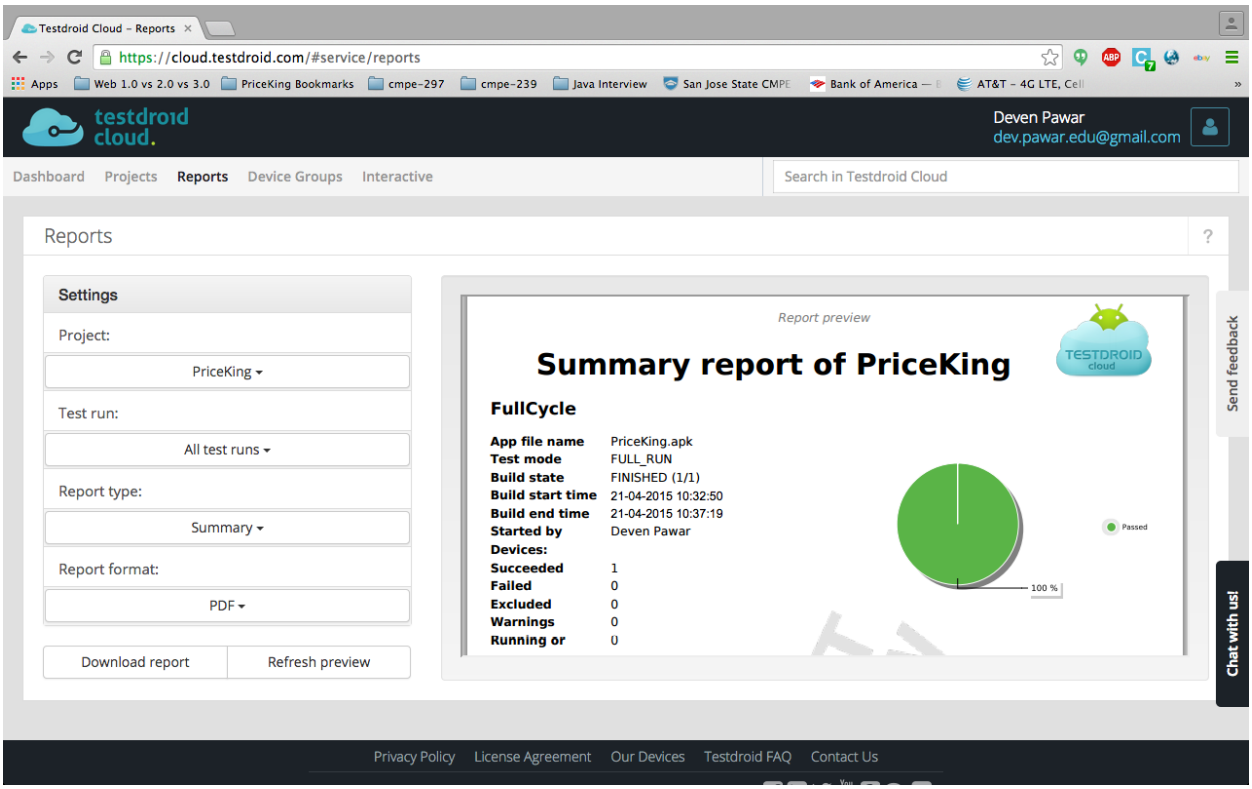
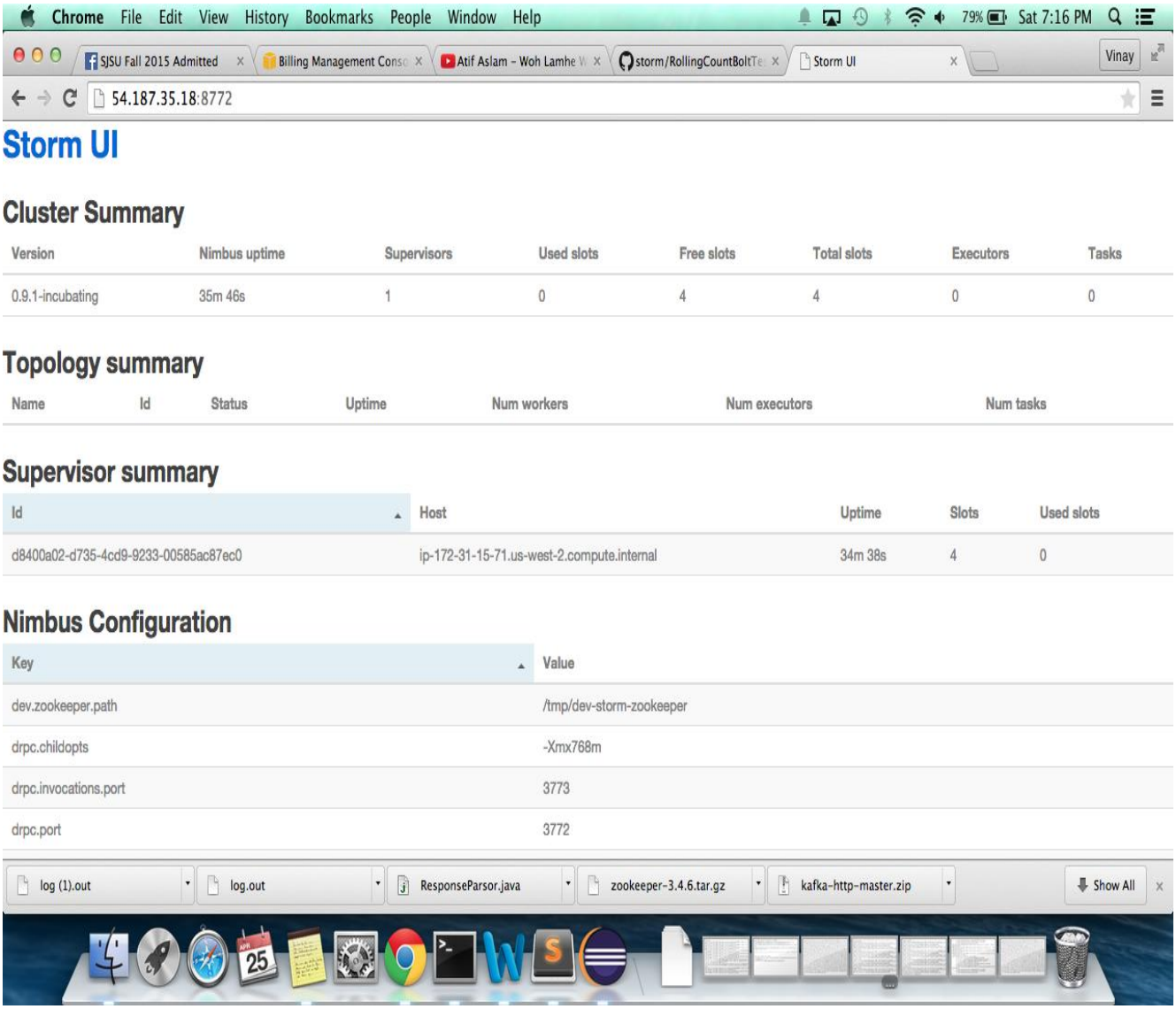


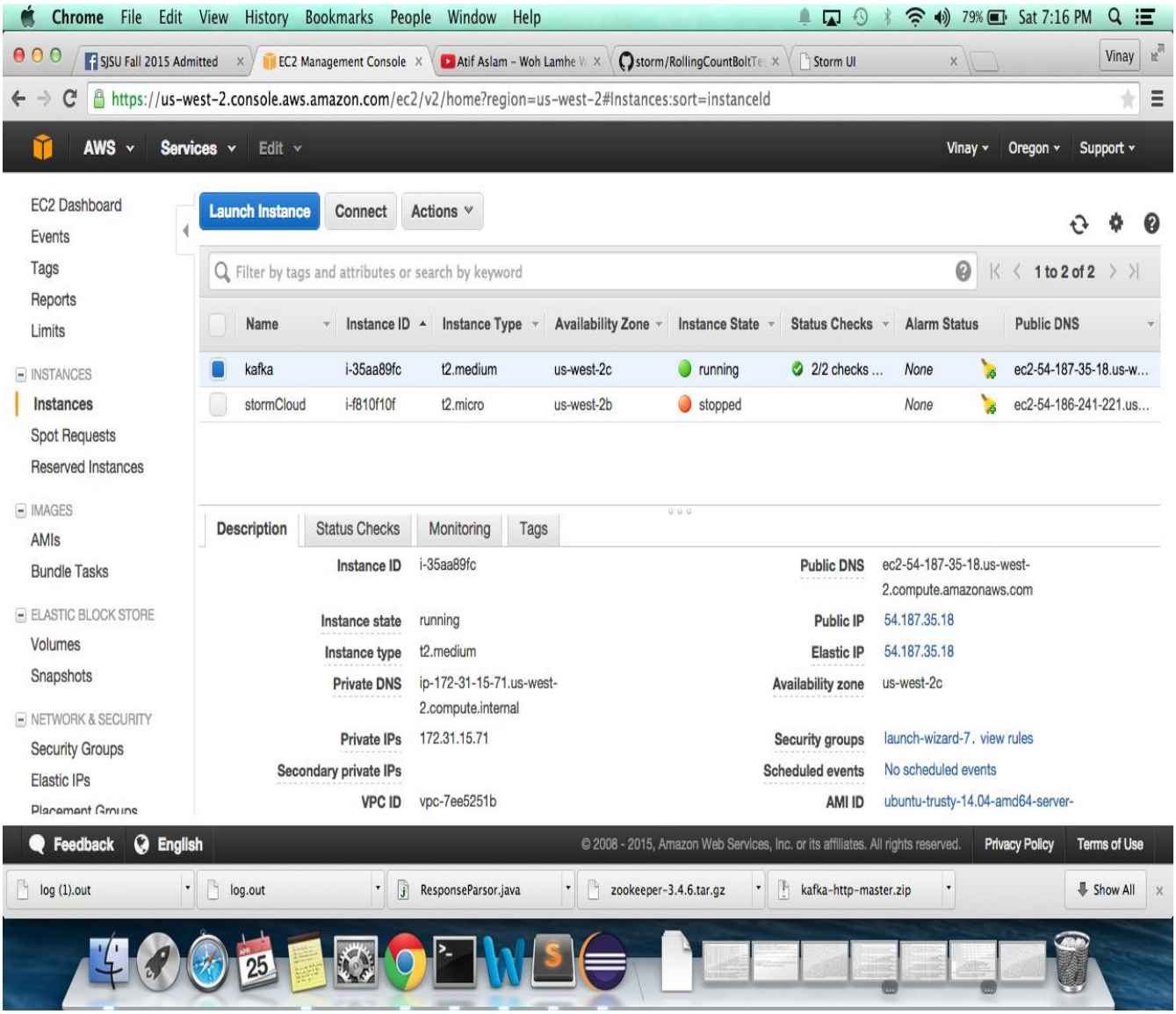
Figure 6.-

Cloud Test Summary



Storm on Cloud


Chrome File Edit View History Bookmarks People Window Help
79% Sat 7:16 PM
SJSU Fall 2015 Admitted
Billing Management Conso
Atif Aslam - Woh Lamhe W
storm/RollingCountBoltTe
Storm UI
Vinay
54.187.35.18:8772
Storm UI
Cluster Summary
Version
Nimbus uptime
Supervisors
Used slots
Free slots
Total slots
Executors
Tasks
0.9.1-incubating
35m 46s
1
0
4
4
0
0
Topology summary
Name
Id
Status
Uptime
Num workers
Num executors
Num tasks
Supervisor summary
Id
Host
Uptime
Slots
Used slots
d8400a02-d735-4cd9-9233-00585ac87ec0
ip-172-31-15-71.us-west-2.compute.internal
34m 38s
4
0
Nimbus Configuration
Key
Value
dev.zookeeper.path
/tmp/dev-storm-zookeeper
drpc.childopts
-Xmx768m
drpc.invocations.port
3773
drpc.port
3772
log (1).out
log.out
ResponseParsor.java
zookeeper-3.4.6.tar.gz
kafka-http-master.zip
Show All


Chrome File Edit View History Bookmarks People Window Help
79% Sat 7:16 PM
SJSU Fall 2015 Admitted
EC2 Management Console
Atif Aslam - Woh Lamhe
storm/RollingCountBoltTe
Storm UI
Vinay
https://us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2#Instances:sort=instanceId
AWS
Services
Edit
Vinay
Oregon
Support
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS
kafka
i-35aa89fc
t2.medium
us-west-2c
running
2/2 checks ...
None
ec2-54-187-35-18.us-w...
stormCloud
i-f810f10f
t2.micro
us-west-2b
stopped
None
ec2-54-186-241-221.us...
Description
Status Checks
Monitoring
Tags
Instance ID
i-35aa89fc
Instance state
running
Instance type
t2.medium
Private DNS
ip-172-31-15-71.us-west-2.compute.internal
Private IPs
172.31.15.71
Secondary private IPs
VPC ID
vpc-7ee5251b
Public DNS
ec2-54-187-35-18.us-west-2.compute.amazonaws.com
Public IP
54.187.35.18
Elastic IP
54.187.35.18
Availability zone
us-west-2c
Security groups
launch-wizard-7. view rules
Scheduled events
No scheduled events
AMI ID
ubuntu-trusty-14.04-amd64-server-
Feedback
English
© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
log (1).out
log.out
ResponseParser.java
zookeeper-3.4.6.tar.gz
kafka-http-master.zip
Show All

6.2.6 Mobile UI Testing

UI testing:

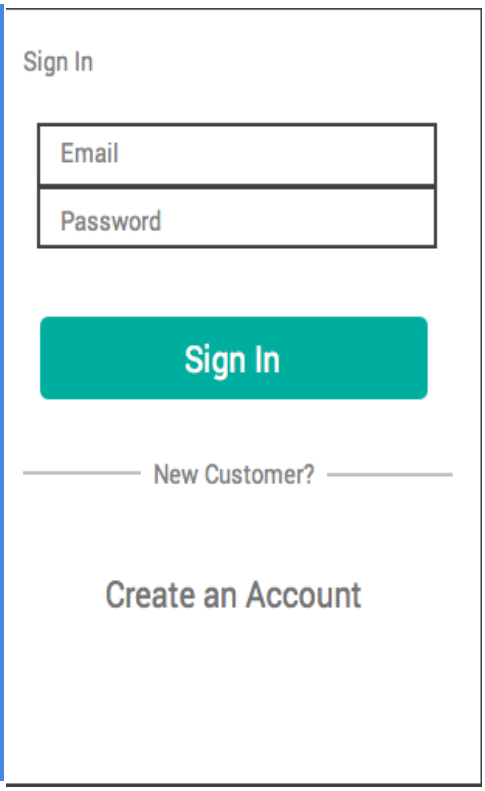
Addition to unit testing of code, it is also necessary for mobile application to check the UI components. As android devices comes in different sizes and shapes in market, it is necessary to check the efficiency of UI that fits into devices. In android, there is a provision for different sizes. In android, all background images, hard coding can set size of font, screens and views within a single screen. This strict coding of XML files and manual handling at runtime helps to maintain the efficiency of UI. For images that are displayed in background have four categories according to size. These categories are ldpi, hdpi, mdpi and xhdpi. For tablet view, images also come in the form of xxhdpi. All images that are displayed in UI are required to be cut accordingly. Moreover, android also provides the display provision in different size of devices at development time. Developer can check whether the display is perfectly fitting into different size of devices or not.

In this project, our team has taken care of these aspects. Our team has cut images according to hdpi, ldpi, mdpi and so on. Moreover, we have done the manual testing approach for testing the UI of app. We created emulators in eclipse of different sizes and checked the UI of app in all these devices. Furthermore, we have also tested out app in different actual devices as eventually app is going to be used by actual users with device and not by emulator. Though there are some tools of UI testing are available in market, we took the manual testing approach for UI testing. This approach is very useful for us as we are running out of time. Moreover we paid more attention for the implementation of functionalities. UI of this app is working perfectly.

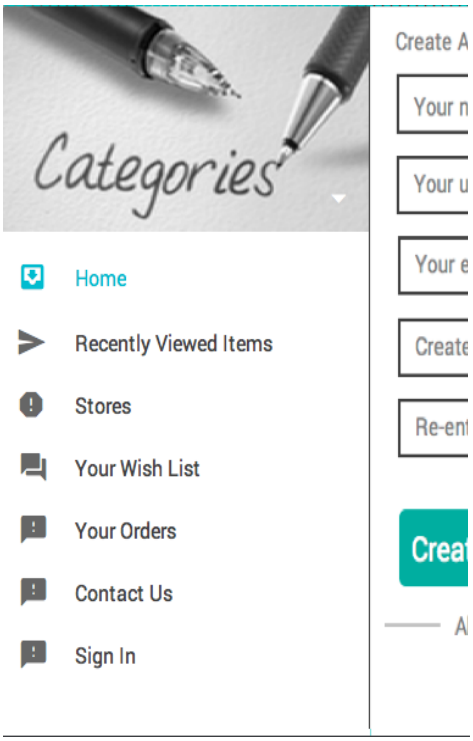
Wireframes:

Wireframes are basically skeletal of 3-dimensional model that is represented by lines and vertices. It uses an image or set of images that represent functional elements of the mobile app in order to decide the structure and functionality of the application. The wireframes we have created for PriceKing Android app are as follows:

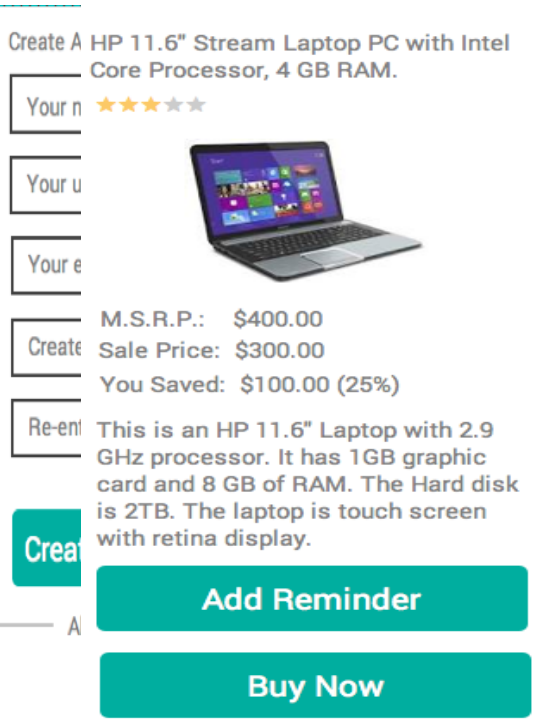
Wireframe 1



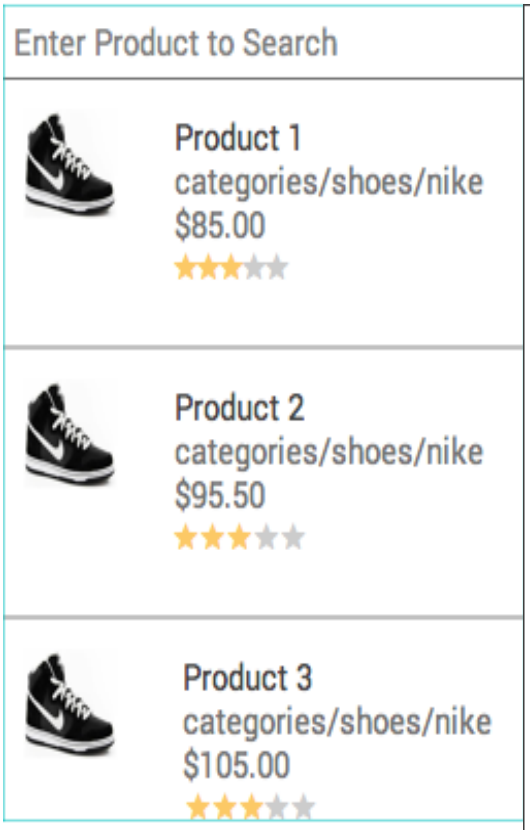
Wireframe 2



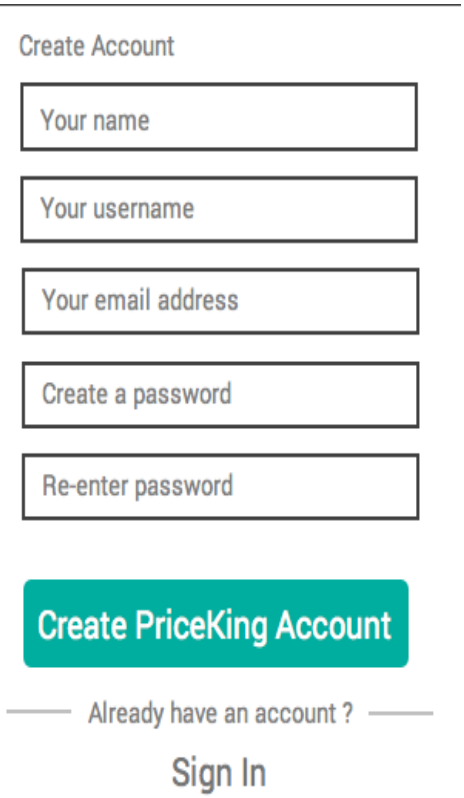
Wireframe 3



Wireframe 4



Wireframe 5



Wireframe 6

Wireframe 7

7. Conclusion and Future work

We feel that this project has been a very good stride for our team in all the ways. It was a challenge and an wonderful opportunity for our team to understand this project from each perspective along with its usefulness in various fields. Demonstrating clear end-to-endrequirements from a Business user domain we provide a complete software plan, design alongwith its development and implementation. To attain full satisfaction from an end user space wetry to test our application in every way for all the components to check the completeness andproper end-to-end functionality for the end user. This project is constantly helped us in learningnew things in each of its software development process. Each of us got a great opportunity tobuild software with lots of learning experience.

7.1 Project Summary

PriceKing is a Mobile App and Web App. After crawling the APIs, the results are filtered and sorted and parsed to the mobile application. Mobile application displays the result along with providing features like add to wish list, recently viewed items, reminder, etc. Web App is targeted for small business users. It gives the platform to the vendors to sell their products and get analytics to improve their Requirements and Analysis.

7.2 Future Work

There are several future enhancements that can be made in our project.Currently only few APIs are taken into consideration for searching best deals but we plan to crawl as many APIs as possible inorder to get a very precise result.To enhance the User Interface of Mobile App as Web as Web App.To add more analytics charts for the small business users

References

[1] Mobile buyers in the United States 2013-2018 | Statistic. (n.d.). Retrieved October 30, 2014, from http://www.statista.com/statistics/241471/number-of-mobile-buyers-in-the-us/

[2] (n.d.). Retrieved October 30, 2014, from http://www.statista.com/topics/1156/coupon-market-trends-in-the-united-states/

[3] IOS Is For Revenue, Android Is For Ads. (n.d.). Retrieved October 30, 2014, from http://www.forbes.com/sites/ayoomojola/2013/09/26/ios-is-for-revenue-android-is-for-ads/

[4] U.S. mobile retail commerce revenue 2013-2018 | Statistic. (n.d.). Retrieved October 30, 2014, from http://www.statista.com/statistics/249855/mobile-retail-commerce-revenue-in-the-united-states/

[5] Comparison shopping website. (2014, October 27). Retrieved October 30, 2014, from http://en.wikipedia.org/wiki/Comparison_shopping_website

[6] (n.d.). Retrieved October 30, 2014, from http://www.bus.indiana.edu/riharbau/RePEc/iuk/wpaper/bepp2011-04-moraga-gonzalez-wildenbeest.pdf

[7] (n.d.). Retrieved October 30, 2014, from http://www.webcredible.com/blog-reports/white-papers/price-comparison.pdf

[8] Fairley, R. (2009). *Managing and leading software projects*. Los Alamitos, CA: IEEE Computer Society

[9] *Systems Engineering Fundamentals.* Defense Acquisition University Press, 2001

[10] Wiegers, Karl E. (2003). *Software Requirements* (2nd ed.). Redmond, WA: Microsoft Press. ISBN 0-7356-1879-8. http://www.processimpact.com.

[11] Laplante, Phil (2009). *Requirements Engineering for Software and Systems* (1st ed.). Redmond, WA: CRC Press. ISBN 1-42006-467-3. http://beta.crcpress.com/product/isbn/9781420064674.

https://storm.apache.org/documentation/Setting-up-a-Storm-cluster.html

http://storm.apache.org/documentation/Running-topologies-on-a-production-cluster.html

https://hadooptips.wordpress.com/2014/05/26/configuring-single-node-storm-cluster/

http://stackoverflow.com/questions/12115160/compiling-jzmq-on-ubuntu

http://zookeeper.apache.org/

http://zeromq.org/

Appendices

Android application has an AndroidManifest.xml file, that is used as a configuration file. If the app requires a new Activity, it has to be specified in this manifest file, else the app will crash throwing “Activity not found” exception. The manifest file also requires permission to access different features of mobile devices such as Camera, Internet, and GPS etc.

AndroidManifest.xml:

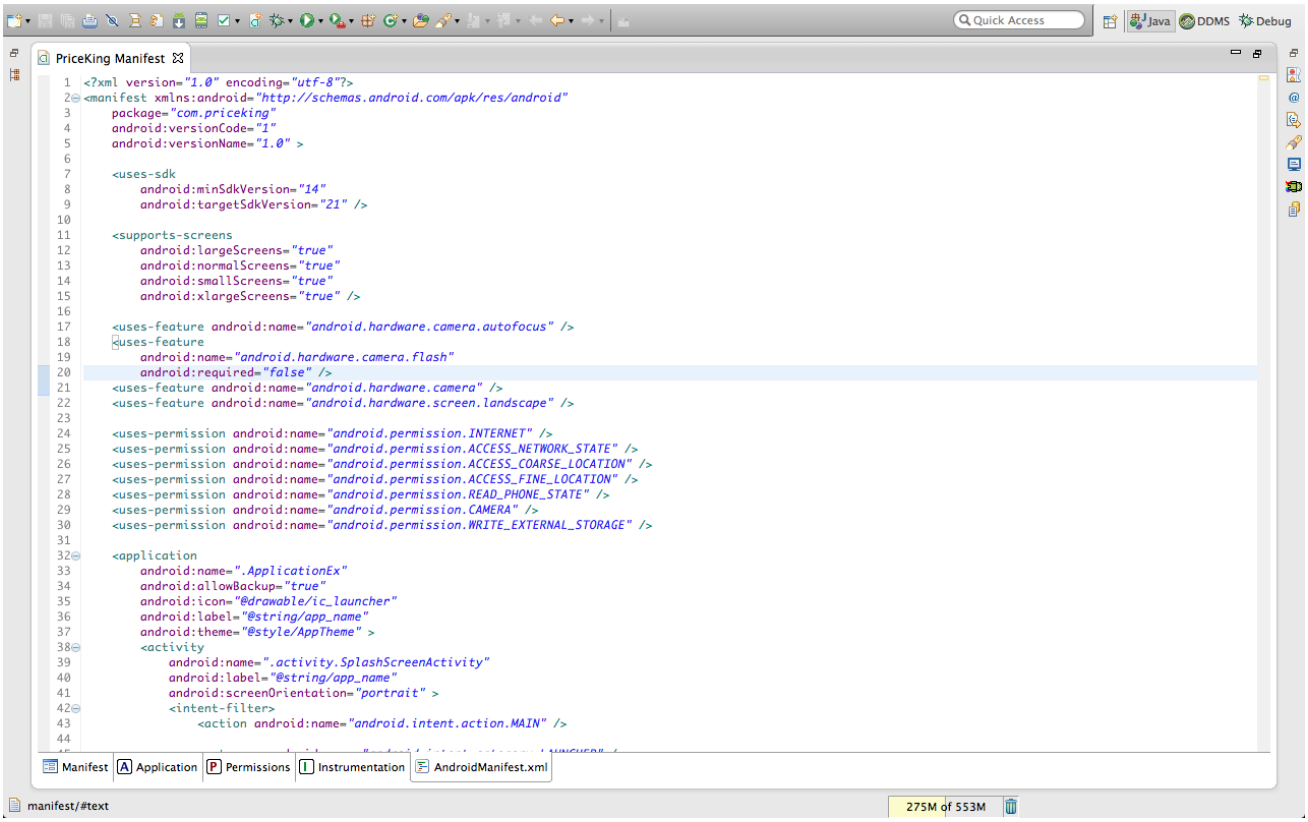


Figure 5.-Android manifest code snippet

```
[{
    "_id": 2,
    "productName": "iPhone6",
    "productDescription": "iPhone6 64 GB Silver",
    "productCategory": "Electronics",
    "thumbnailImage": "",
    "productUrl": "N/A",
    "price": "699",
    "uploadedBy": "hardik",
    "isDeleted": false
}, {
    "_id": 3,
    "productName": "Bed Frame",
    "productDescription": "This is best price in the market.",
    "productCategory": "Furniture",
    "thumbnailImage": "",
    "productUrl": "N/A",
    "price": "40",
    "uploadedBy": "hardik",
    "isDeleted": false
}]
```

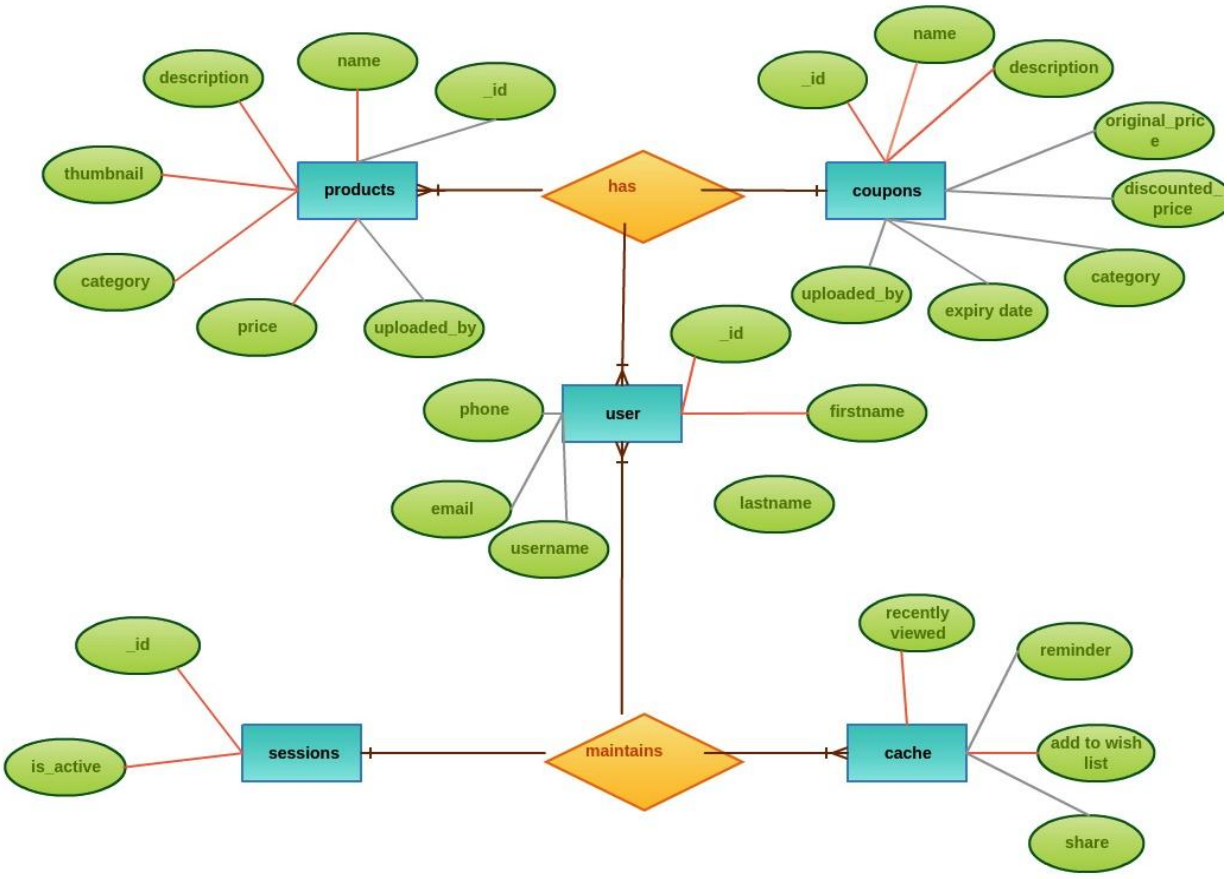
```
{
    "errCode": 1,
    "errorMessage": "Incorrect Password",
    "uid": "hardik"
}
```

```
{
    "errCode": 0,
    "errorMessage": "success",
    "uid": "hardik",
    "email": "hardikjoshi.se@gmail.com"
}
```

```
{
    "errCode": 1,
    "errorMessage": "Incorrect Username",
    "uid": "hardik1"
}
```



```
[{
    "_id": 2,
    "name": "iPhone 6 Case",
    "description": "This offer is for limited time. Hurry Up! - 50% off on eBay",
    "originalPrice": 20.0,
    "discountedPrice": 10.0,
    "couponCode": "HGY87RIO",
    "expiryDate": "May 9, 2015 7:00:00 AM",
    "uploadedBy": "hardik",
    "isDeleted": false
}]
```



ER Diagram for Priceking