PROJECT REPORT

PriceWise

Master of Software System

CICS 530

(ADVANCED SOFTWARE ENGINEERING PROJECT)

Submitted by:

Group 6

Contents

Introduction

The “Price Wise” is a web application used to compare product price and features from different e-commerce websites based on user needs. This project is going to affect all users who wish to buy quality products on low price. Normally ‘price comparison’ web applications don’t include features such as wish-list, history, daily/weekly deals etc. We have planned to target these needs and allow a certain level of freedom to users. Our solution application contains a few other important features (explained in Product backlog document) to make the comparison easier.

We are following Agile technique for project development. Our sprints would be bi-weekly with estimates by days. A team of four will work on all user stories.

1.1 Team Product Backlog (please refer to attached excel sheet)
1.2 Team Sprint Backlog (Please refer to attached excel sheet)

Product Vision

The “PriceWise” project is a web application that is used to compare product prices and features from different websites.

Motivation/Opportunity

As explained earlier, this project will compare prices from different websites as per user requirement. We decided not to buy something off-the-shelf because most of the products in market do not have all the features in one application. We have decided to let the users choose the websites they want to compare from. Users can chose one either one or both websites to compare products.

Problem Statement:

The problem of	Comparing prices of products on different ecommerce websites
Affects	All web users
The impact of which of	Side by side Price and product comparison
A successful solution would be	A web application that makes comparison of price and product easy and user-friendly

Product Position Statement:

For	All chrome users
Who	online shoppers
Our System	is all software
That	Compares the price
Unlike	Other products that search default websites only
Our Product	Is user friendly and runnable at customer site

Users: The team aims to address online shoppers who wish to compare prices from different websites before buying any product. The users are supposed to have basic working knowledge of computer and Internet.

Feature List: 'PriceWise' is determined to have following features:

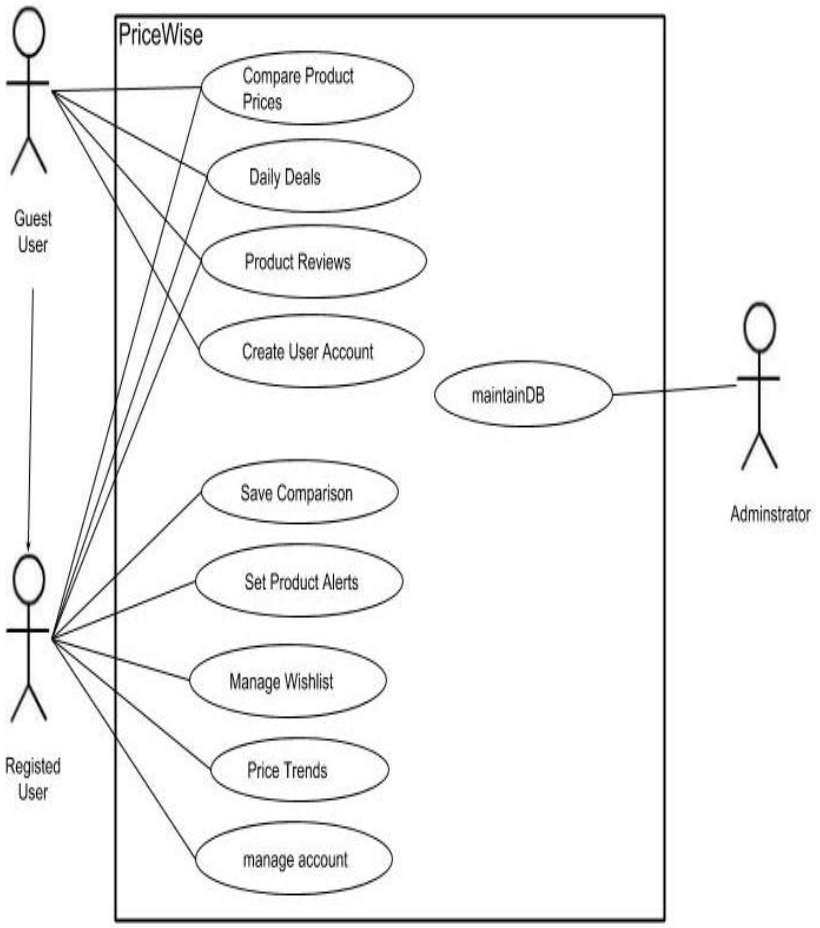
- Compare Product
- Wish List
- Price Alerts
- Daily Deals
- Save Comparison
- Price Trends
- Reviews

Constraints**:** 'PriceWise' requires any web browser to operate. The team has agreed to use Relational database because we need to store the account details and product details for further features. Browser friendly programming languages like HTML, CSS, and PHP will be used for development under MVC architecture. Open source libraries like Swift will be used wherever needed.

Scope and Limitations**:** As explained in feature list 'PriceWise' will give users an option to choose websites to compare products from. But the team has decided to use only 2 major websites. Though more websites will be added in future. Also 'PriceWise' will currently compare prices only from Canadian websites.

Assumptions and Dependencies: Our only dependency presently is proper working of web browsers and the API calls made on different websites.

Use Cases: High Level Use Case:



Functional Requirements

This web application supports three types of actors

Guest user / unregistered user should be able to

- Input the product in the textbox
- Retrieve the price list of various merchants for the product
- Select the merchant name to get the coupon updates
- Get the daily deals merchant wise

Registered user must be able to

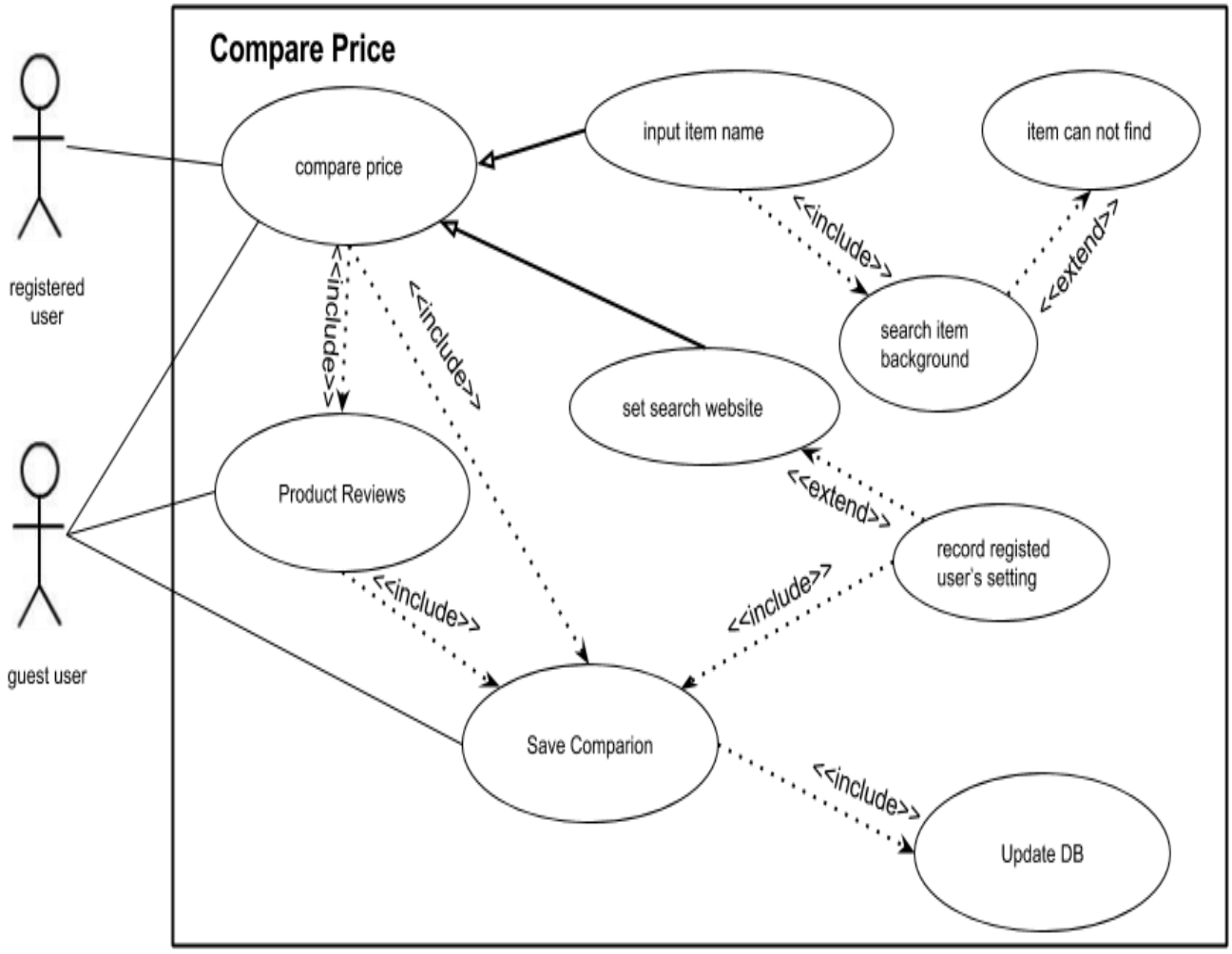
- Input the product in the textbox

- Retrieve the price list of various merchants for the product
- Login into the account/ Manage account
- Add/remove product to the wish list
- Manage alerts
- Save the comparison
- Get the price trends

The *system administrator* must be able to access the user database details and add, remove or change the details any time.

Sub Level Use Cases:

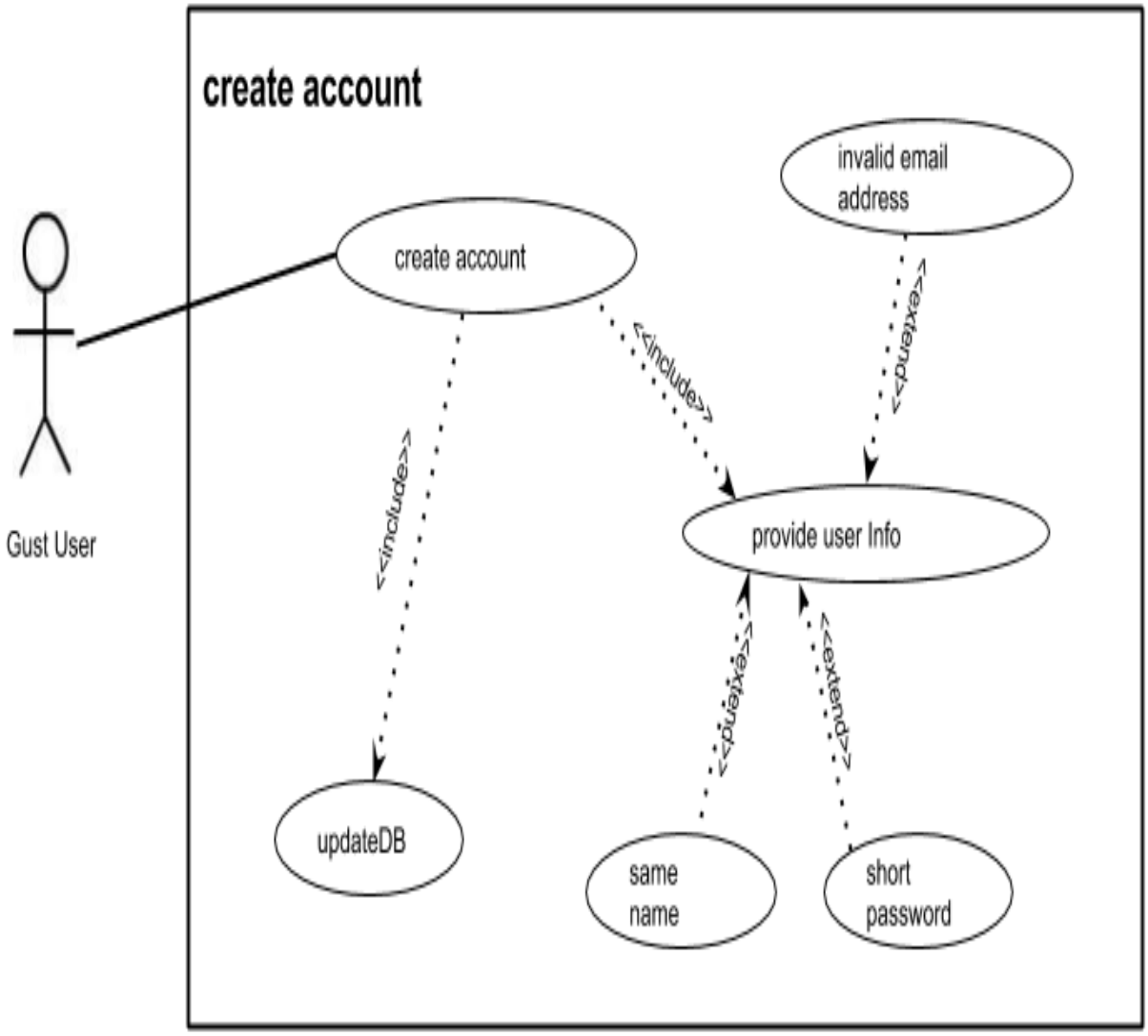
1. Compare Prices:



Textual description: ***Use Case: Product Selection and Compare Price***

Use Case ID	1.0	
Use Case Name:	Compare Price	
Participating Actors:	Guest User , Registered User	
Description:	User wants to buy cheap products online	
Pre conditions	User must open the web browser. User internet connection should be working User has access to the server website	
Post conditions	User must be able to successfully compare the product prices	
Flow of Events:	**Step**	**Action Description**
	1	User can login into the application or can directly open the google chrome browser and enters into specified website to look for online product
	2	'PriceWise' application fetches the product data from various websites
	3	'PriceWise' application compares the product prices and display the chart to display the price trend of the product during past few days
	4	User selects the from product available and registered user is allowed to save the comparison
Exceptions	**Item #**	**Action Description**
	1	Internet stop working
	2	Product is not available on website
	3	Username and password does not match

2. Create Account

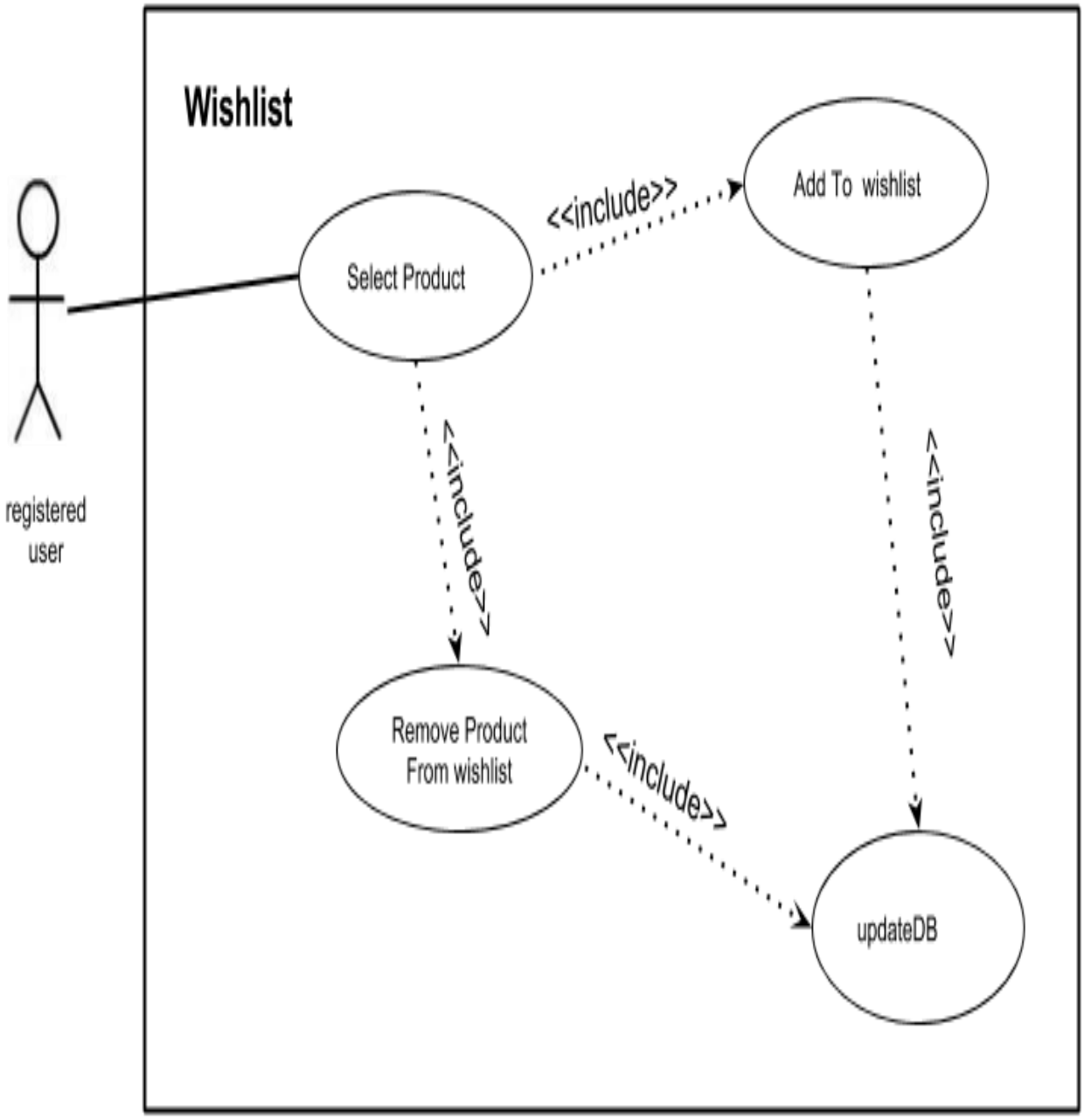


Textual description:

Use Case: Create Account

Use Case ID	2.0	
Use Case Name:	Create Account	
Participating Actors:	Guest User	
Description:	User has a request to create account to access the additional features	
Pre conditions	User must open the web browser. User internet connection should be working User has access to the server website	
Post conditions	User must be able to successfully create the account	
Flow of Events:	**Step**	**Action Description**
	1	User click on the “Create Account “ button
	2	User enters the name, email address and the password.
	3	New account is created and the database is updated.
Exceptions	**Item #**	**Action Description**
	1	Internet stop working
	2	Email ID already exists in the database

3. Manage Wish list

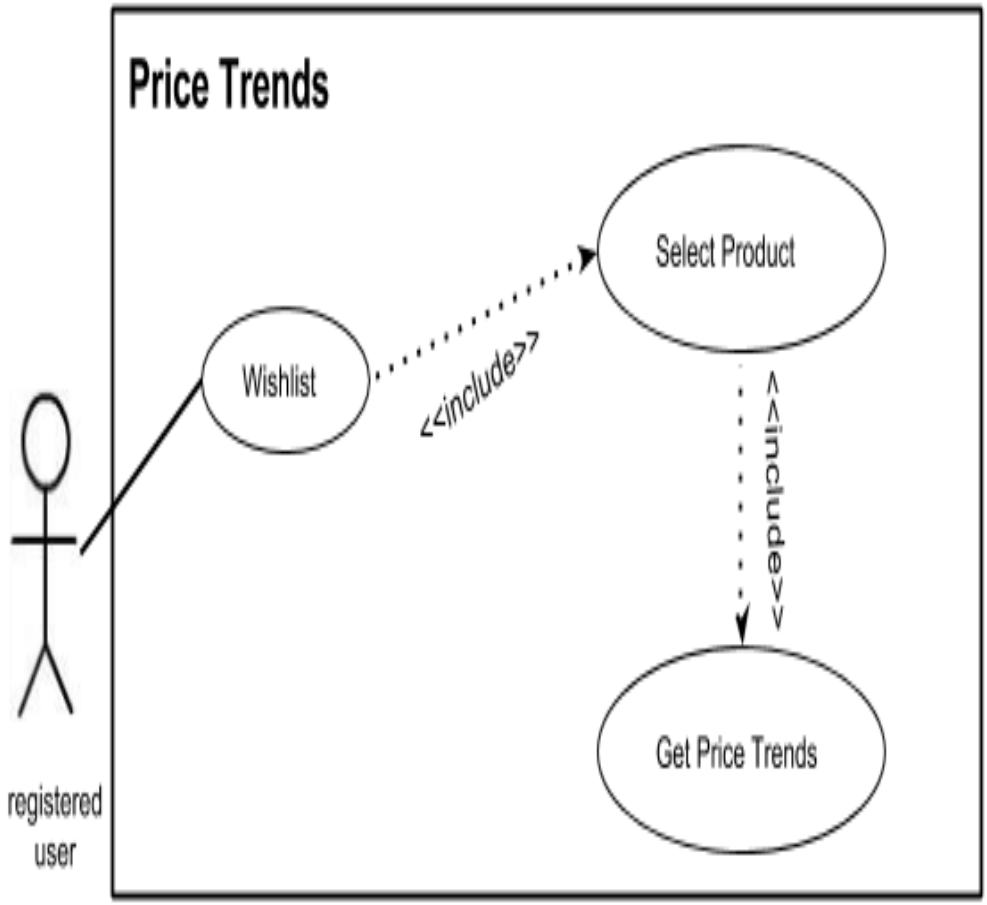


Textual description:

Use Case: Manage Wishlist

Use Case ID	3.0	
Use Case Name:	Manage Wishlist	
Participating Actors:	Registered User	
Description:	User has a request to add product to wishlist to track the product for future.	
Pre conditions	User must open the web browser. User internet connection should be working User has access to the server website	
Post conditions	User must be able to successfully login into the account	
Flow of Events:	**Step**	**Action Description**
	1	User click on the Select Product button to add new product to the wish list. Press on the star to add to wish list.
	2	User can also click on the “Wishlist” tab to delete the existing product form the wish list
	3	Update the database.
Exceptions	**Item #**	**Action Description**
	1	Internet stop working
	2	Product already in the database.

4. **Price Trends**



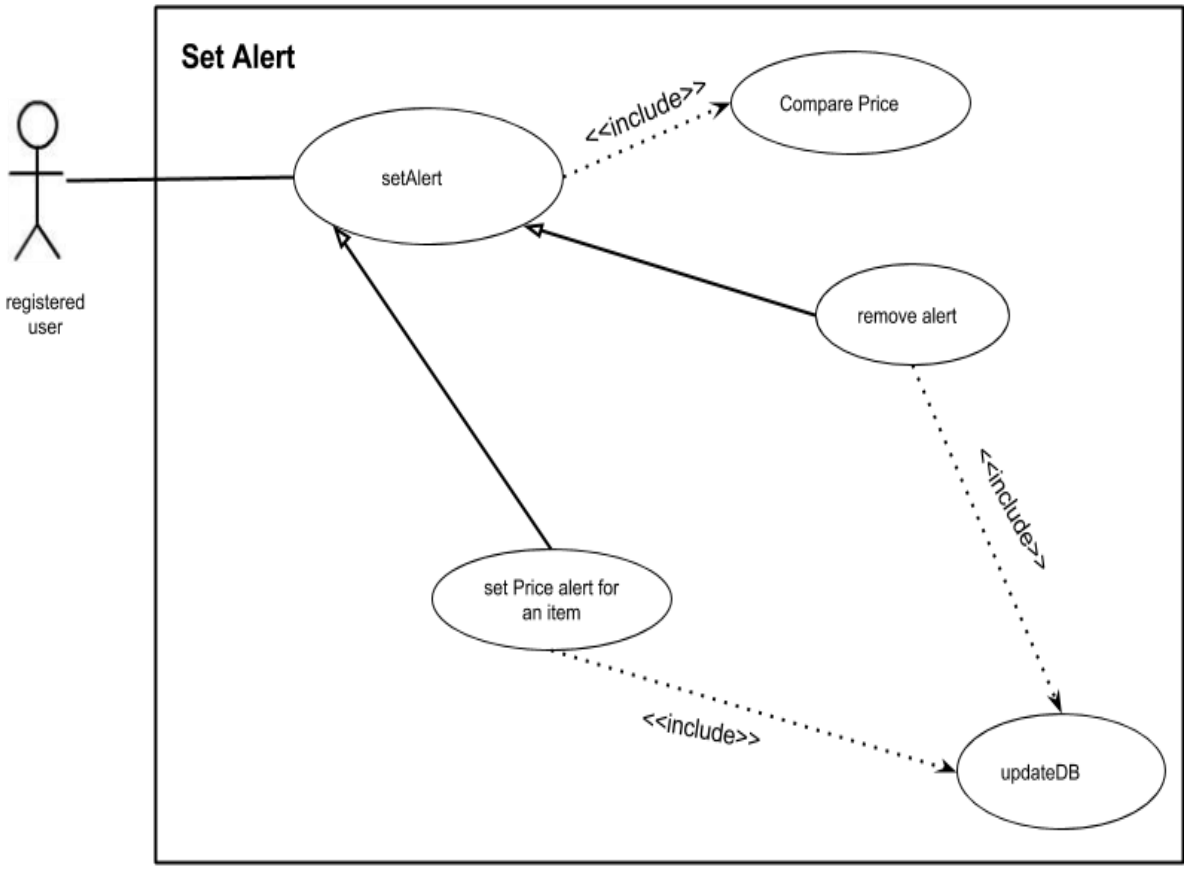
Textual description:

Use Case: Price trends

Use Case ID	4.0
Use Case Name:	Price Trends
Participating Actors:	Registered User
Description:	User has a request to get the price trends of the product saved in the wishlist

Pre conditions	User must open the web browser. User internet connection should be working User has access to the server website	
Post conditions	User must be able to successfully create a new alert or update the existing alerts	
Flow of Events:	**Step**	**Action Description**
	1	User logins as registered user into the PriceWise.
	2	Click on the wishlist
	3	Click on the price trends to get the price variations in the product in last 15 days.
Exceptions	**Item #**	**Action Description**
	1	Internet stop working

5. Set Alerts



Textual description:

Use Case: Set Alert

Use Case ID	5.0
Use Case Name:	Set Alert
Participating Actors:	Registered User

Description:	User has a request to create alerts or update alerts. The user gets and email alert when the price of the product drops.	
Pre conditions	User must open the web browser Google Chrome. User internet connection should be working User has access to the server website	
Post conditions	User must be able to successfully create a new alert or update the existing alerts	
Flow of Events:	**Step**	**Action Description**
	1	User logins as registered user into the PriceWise.
	2	User search and compare the products and click on the “Set Alerts “ button
	3	The script will run twice a day to check if there is any price drop in the selected product price
	4	After the selection of the alerts, database is updated.
Exceptions	**Item #**	**Action Description**
	1	Internet stop working

Non-Functional Requirements

1. **Usability**

- The registered user should be able to manage his/her account without any training.
- All users should be able to use basic functions including compare price, check reviews, and check coupons showed in the main application interface without any training.
- The application should also provide a user manual for different users.

2. **Reliability**

- The application shall not leave any cookies on the registered users' computers containing their password.
- The application should not lose data in case of power failure.
- With or without a user account, users should both be able to use basic functions including compare price, check reviews, and check coupons showed in the main application interface.

3. **Performance**

- The application should take initial load time depends on Internet connection strength.
- The application retrieve data in less than 30 seconds.
- The application should handle 1000 users at the same time.

4. **Supportability**

- The application should be run in Chrome in both Microsoft Windows, Mac OS X and Linux operating system.

5. **Implementation**

- The application should be based on web and to be run from Chrome browser.

6. **Security**

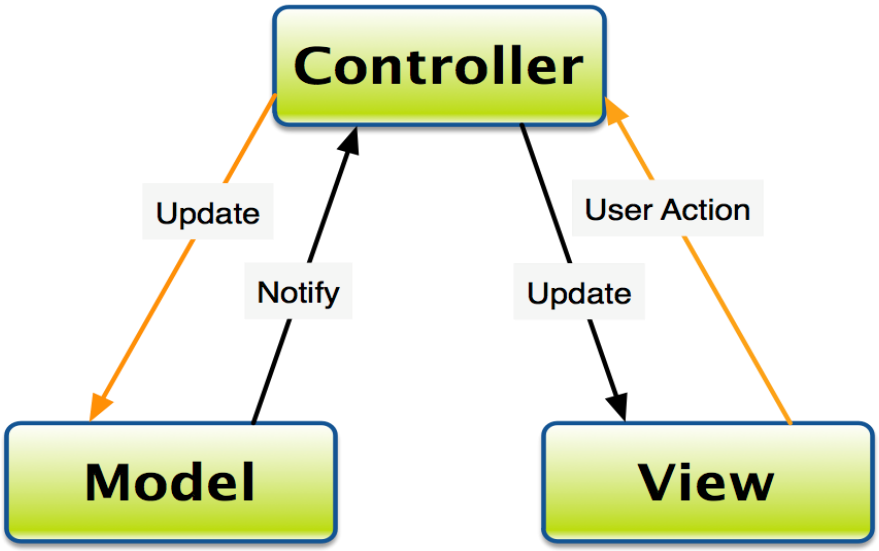
- The application should maintain all the registered user information in encrypted form.
- The application shall automatically log out all registered users after a period of inactivity.

Design and Architecture

Architecture and Rationale

The team is using MVC to develop the project. Model–view–controller (MVC) is a software architectural pattern mostly for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. MVC offers architectural benefits over standard JavaScript — it helps you write better organized, and therefore more maintainable code. This pattern has been used and extensively tested over multiple languages and generations of programmers. The benefits of using MVC patterns are:

1. **Separation of concerns:** The separation the three components, allows the re-use of the business logic across applications and Multiple User Interfaces can be developed without concerning codebase.
2. **Developer specialization and focus:** The developers of UI can focus exclusively on the UI screens without bogged down with business logic. The developer of model/business can focus exclusively on the business logic implementations, modifications, updations without concerning the look and feel and it has nothing to with business logic.
3. **Parallel development by separate teams:** Business logic developers can build the classes, while the UI developers can involve in designing UI screens simultaneously, resulting the interdependency issues and time conservation. UI updations can be made without slowing down the business logic process. Business logic rules changes are very less that needs the revision/updations of the UI.

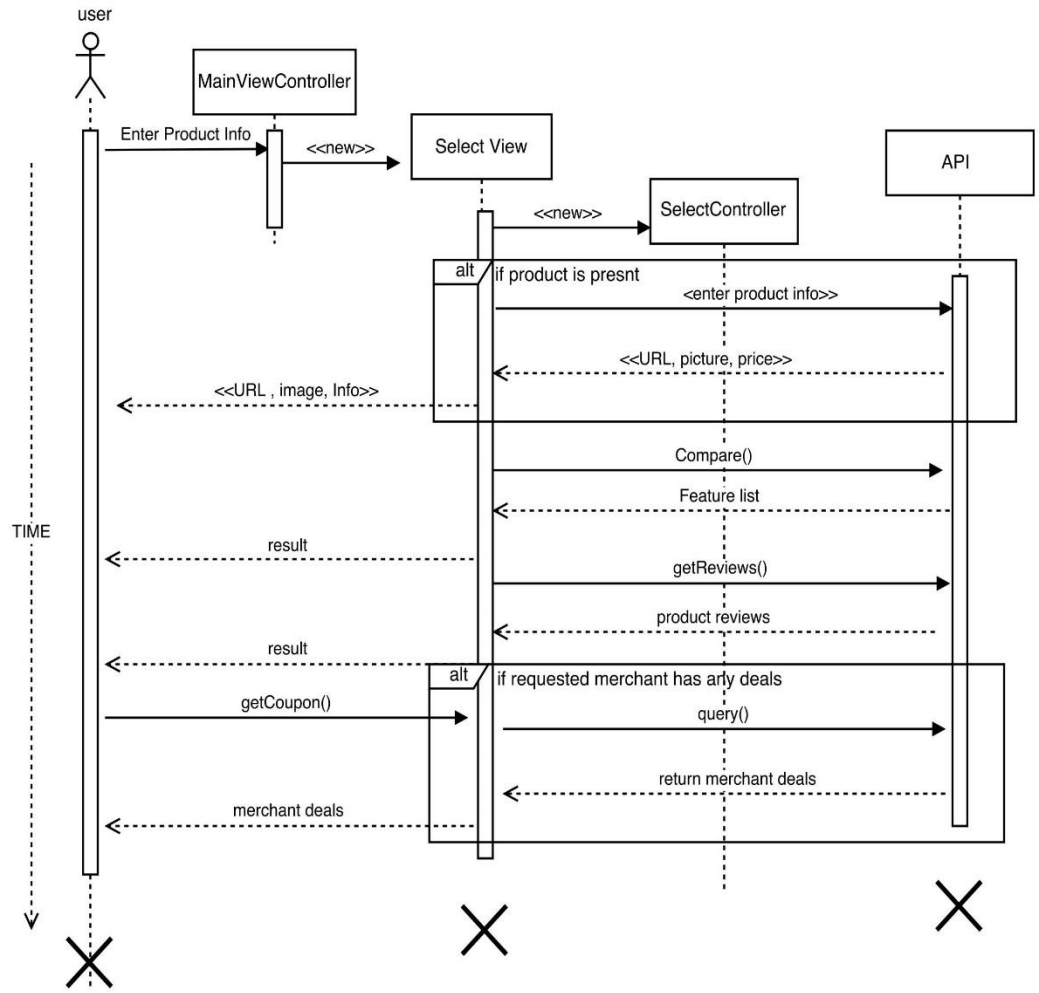


Model – View – Controller

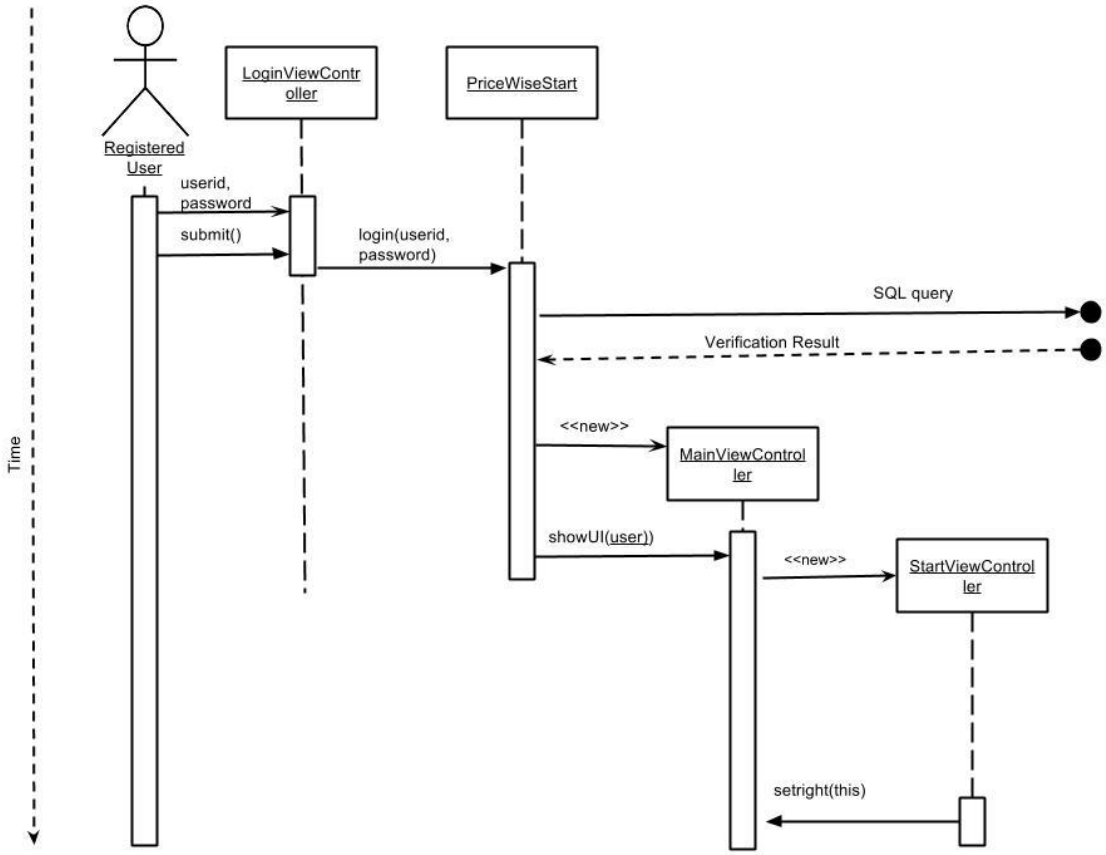
The architecture is based on 4 + 1 approach. The details are:

Logical view: The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view are following sequence diagrams:

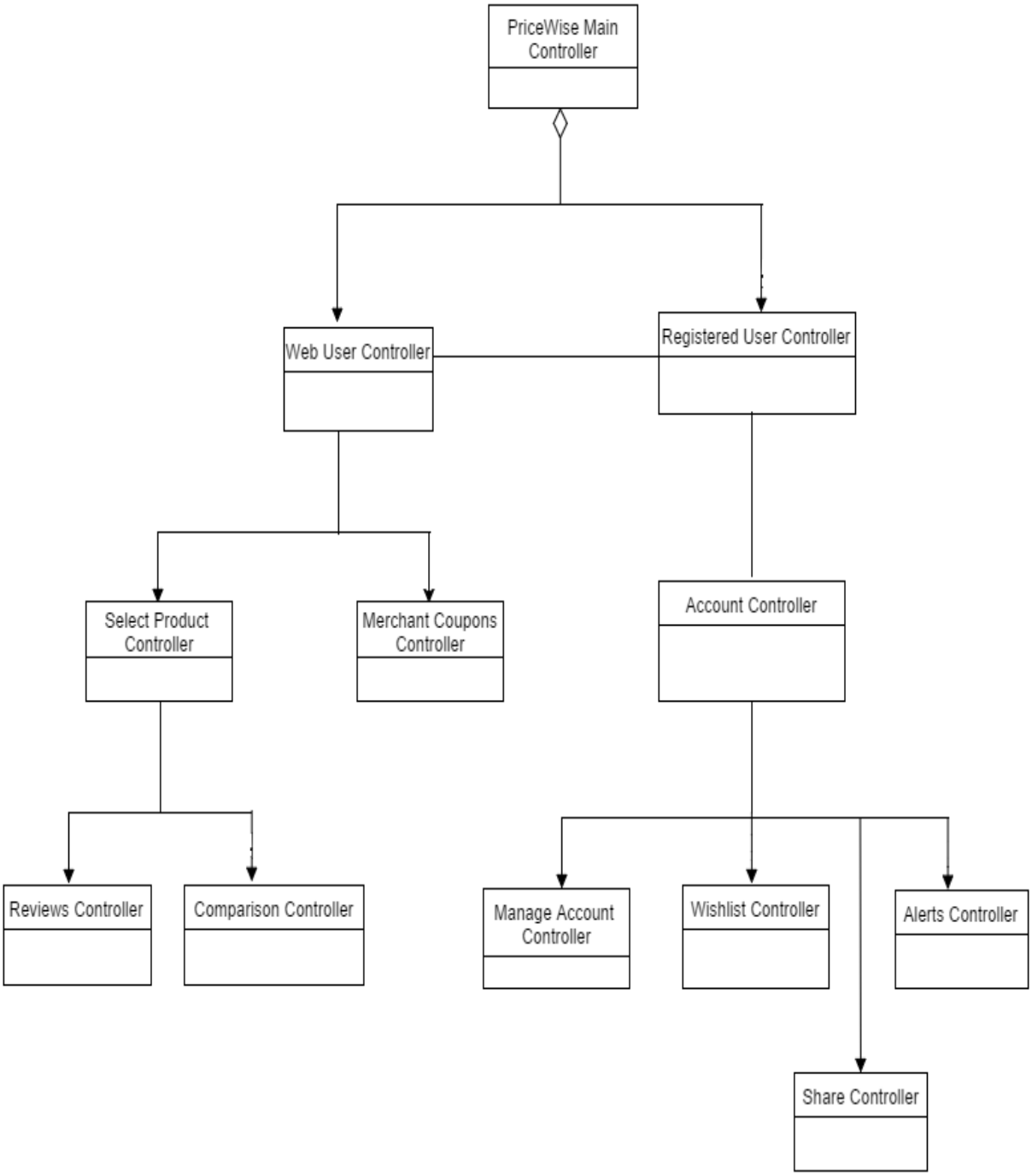
i) Select product and do comparison - Sequence Diagram Description: The flow starts when 1) the actor User enters the product keyword 2) the application sends API call to website 3) validate if product is present 4) API returns the object with the required information detail of the product. 5) Now the user can select from two options- Compare and Get Reviews 6) if user selects compare option then another API call is made to get the feature comparison list 7) if user selects the review option only the reviews are returned using the API call for the selected products. 8) New flow can start when user opts for the "get Coupon option" from the start menu. 9) Validate if required merchant has coupons 10) API call return the coupon list for the desired merchant. 11) On successful API call return user has either done the comparison / read the product reviews / or retrieved the desired merchant coupons.



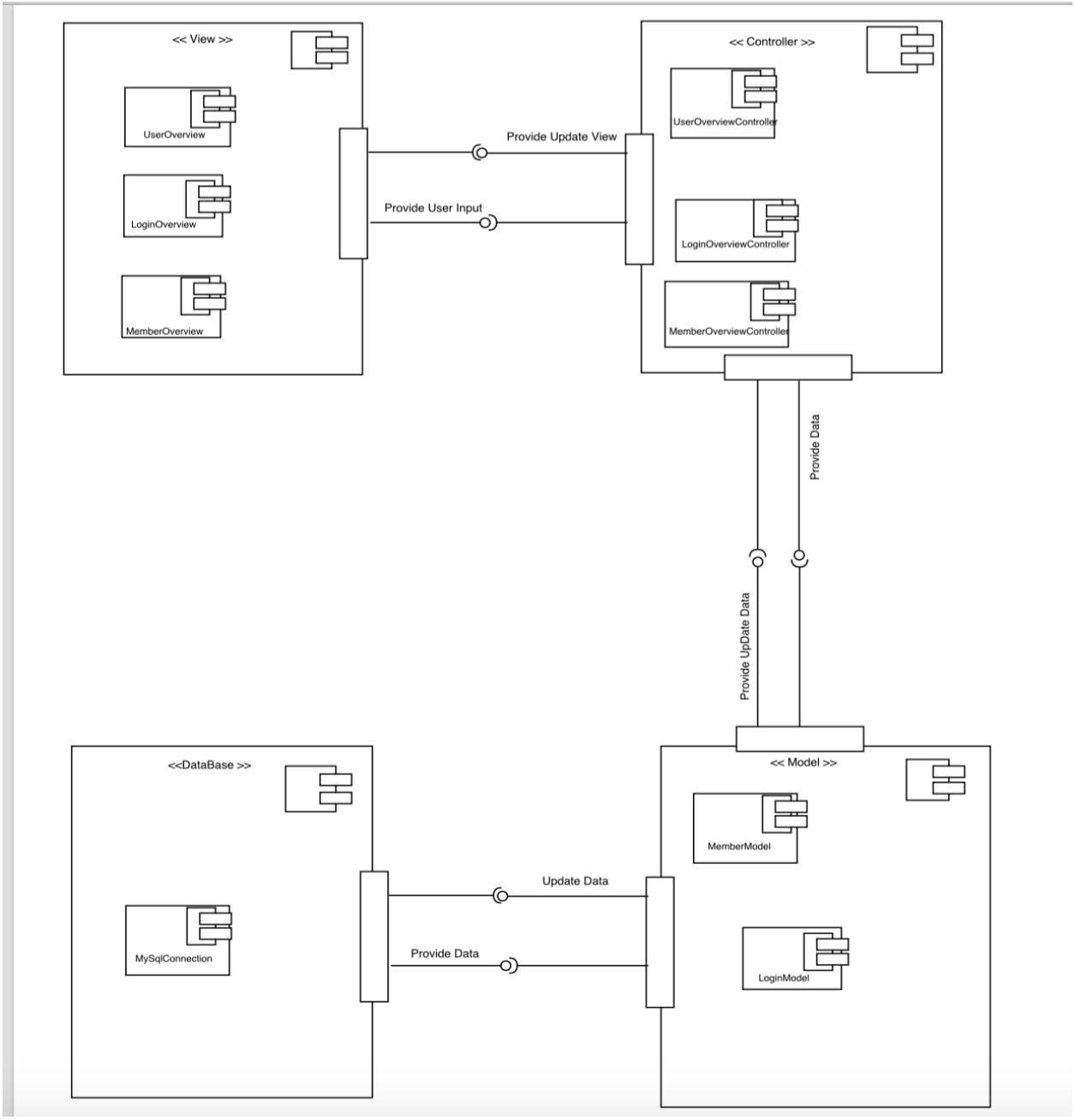
ii) Login: Sequence Diagram Description: The Flow starts when 1) the actor "user" enters the userid and password 2) validate the information from the database 3) Database validates the user information 4) if user is not present in the database then ask the user to sign up 4) if user information is valid then user is able to go into the user account home page. 5) On successful connection to the server database, the user must be able to view the user details.


Time
Registered User
LoginViewController
PriceWiseStart
userid, password
submit()
login(userid, password)
SQL query
Verification Result
<<new>>
MainViewController
showUI(user))
<<new>>
StartViewController
setright(this)

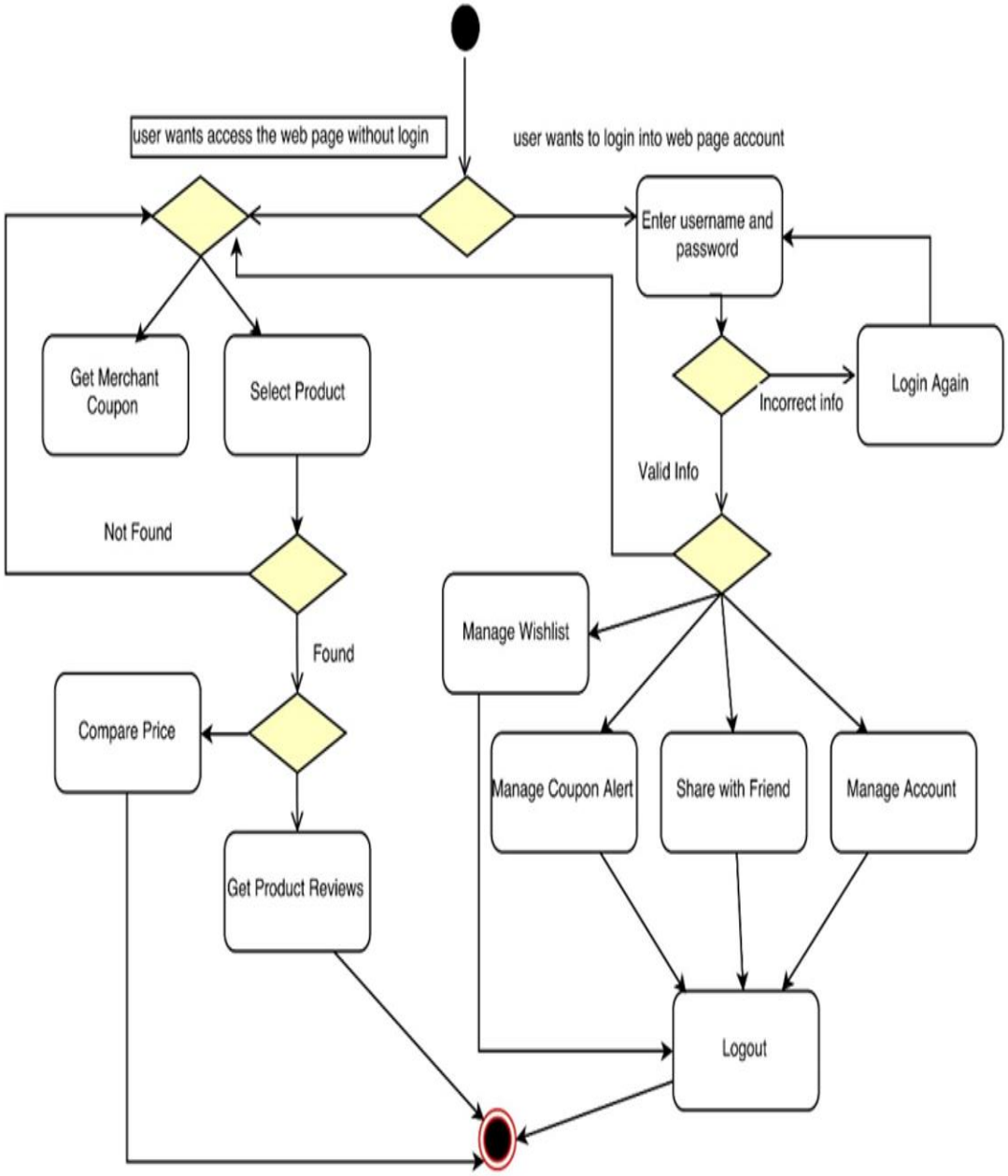
Class diagram:



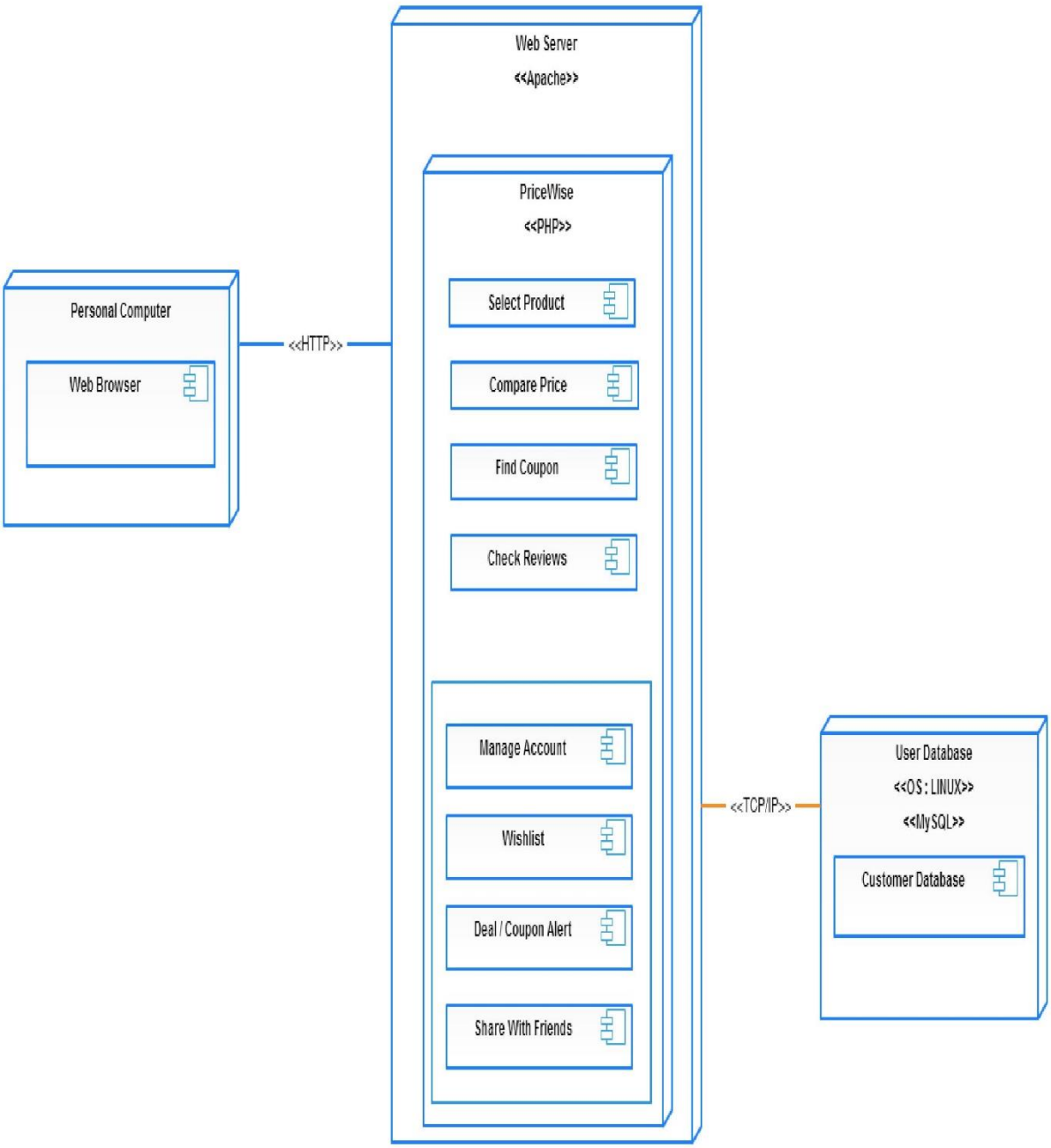
Development View**:** The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. UML **Component diagram** is used to describe system components:



Process view: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the **Activity diagram**:



Physical view: The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the **Deployment diagram**:



Scenarios: The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as use case view.

Data**:** We are using relational database to manipulate the user data. Below is the table design and E-R diagram to explain this:

ER diagrams:

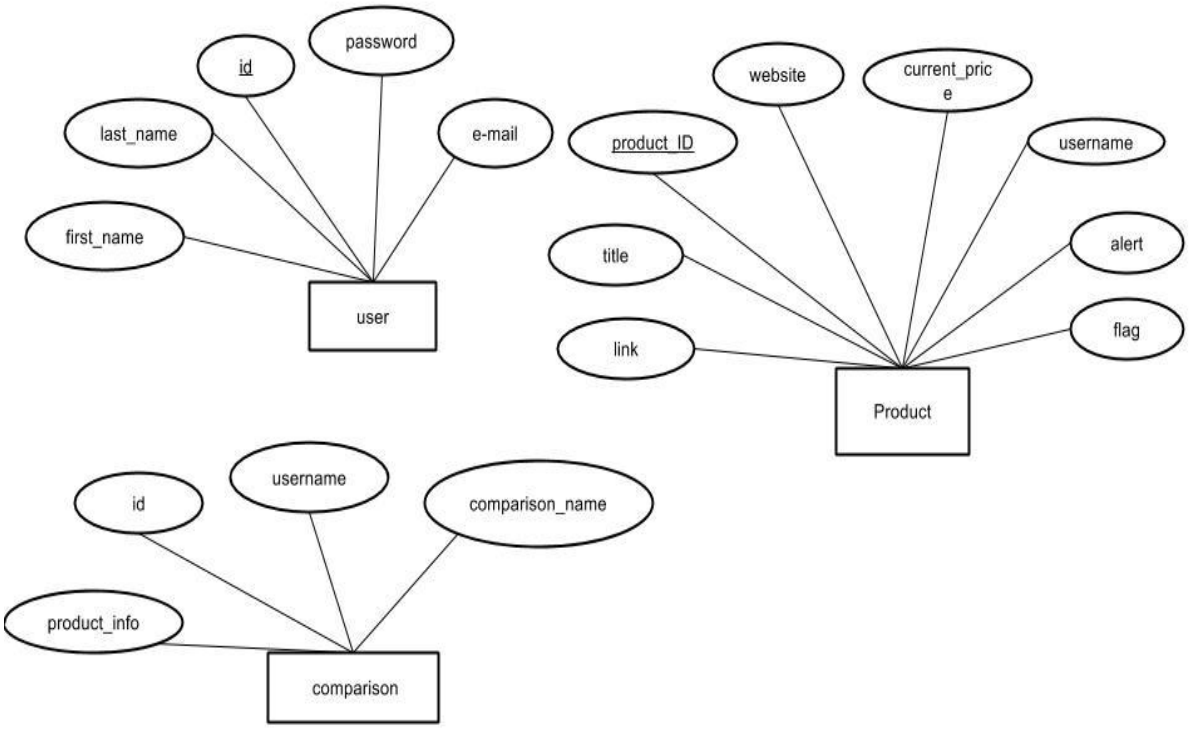
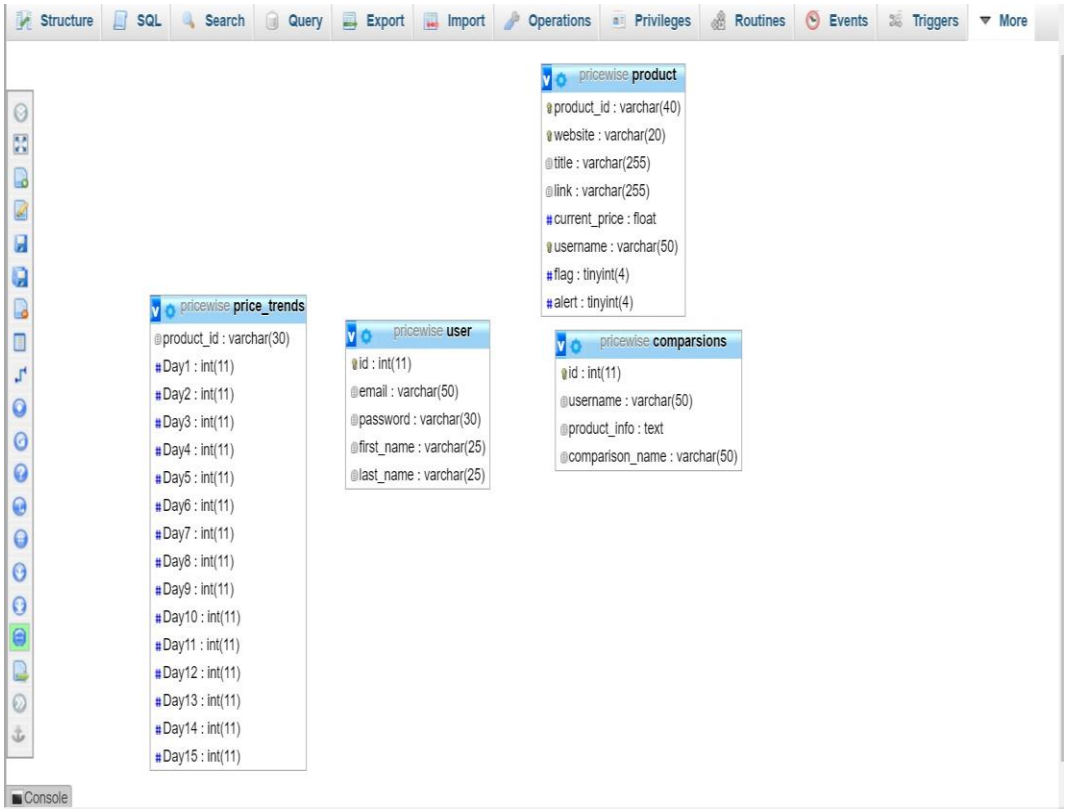
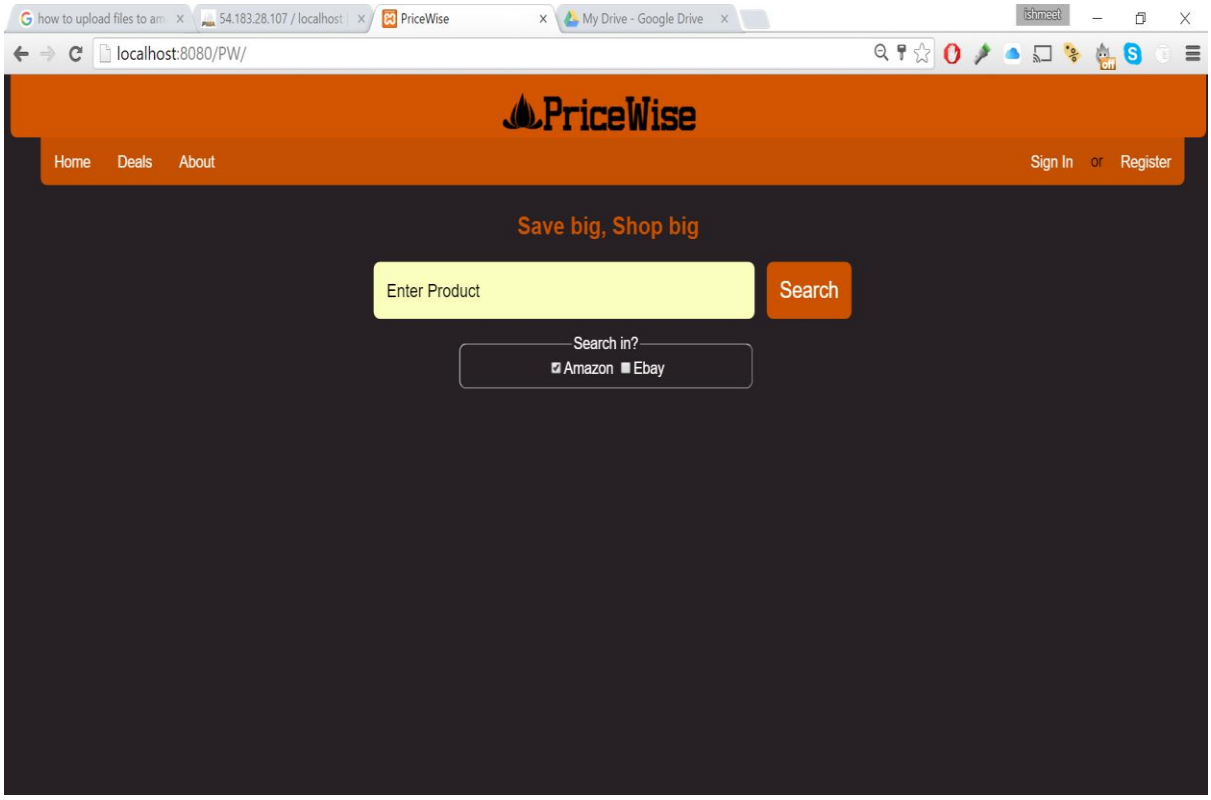


Table Design

Database Schema Design:



User Interface



Test Plan

Objective of Test Plan document

This test plan will serve three main objectives:

1. **Describe our testing approach for the subsystems of the project:** This document will describe the testing approach on the system's components, the features these components will support and the criteria that will determine if the feature is complete or not.
2. **Define testing responsibilities and team work flow:** This document will serve as a reference to define responsibilities for team members in ensuring product quality. The specific testing efforts required for the team roles will be defined.
3. **To serve as a basis for integrating the project:** This document will describe how the project will be integrated.

Verification Strategy

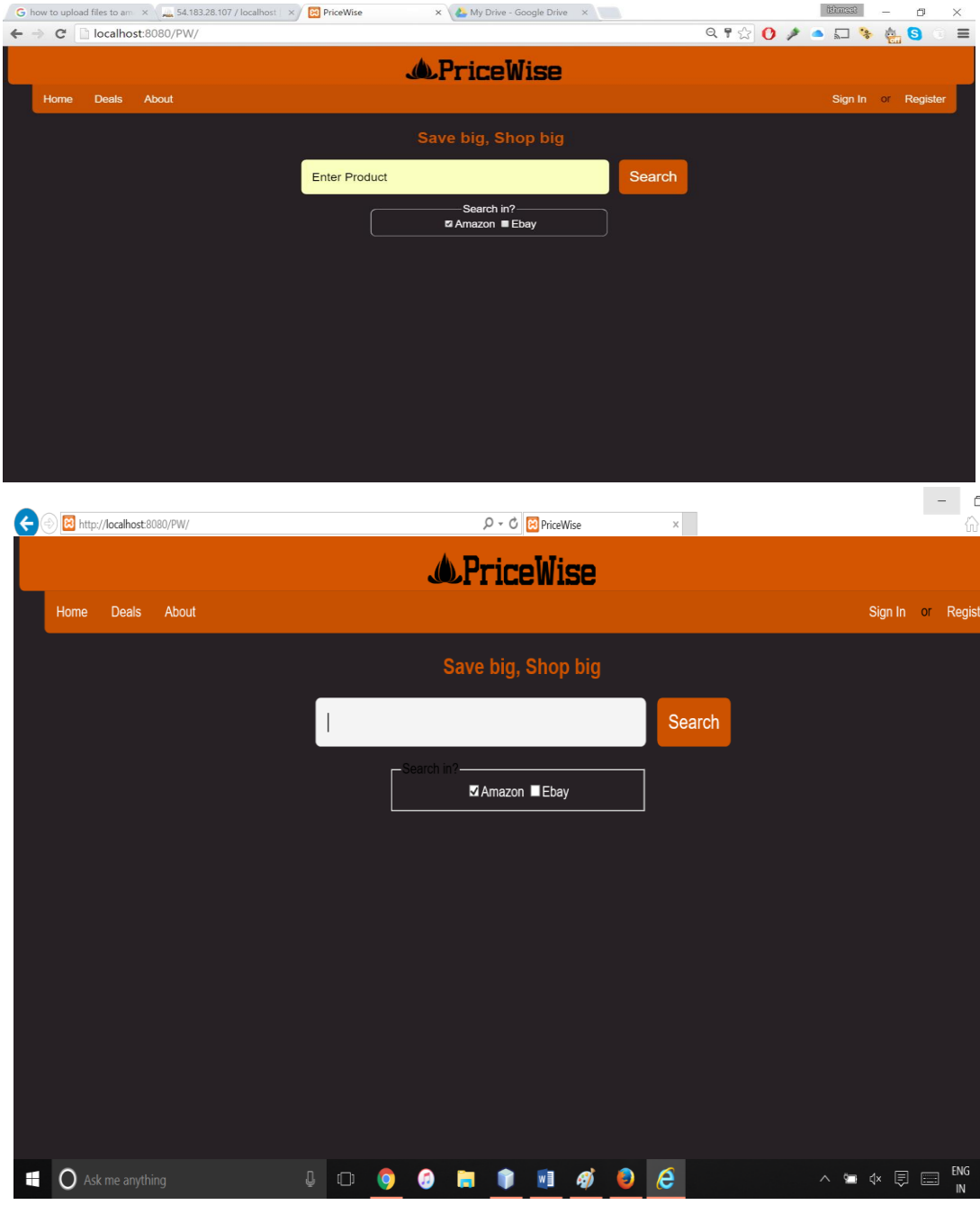
At the initial stage of project we provided the user with the requirement analysis document to validate the user requirements. After the deep review and few meetings we came across the

design document which is reviewed by the user. Also at every stage of development we will show our project demo to the user to get the feedback to verify the righteousness of our design.

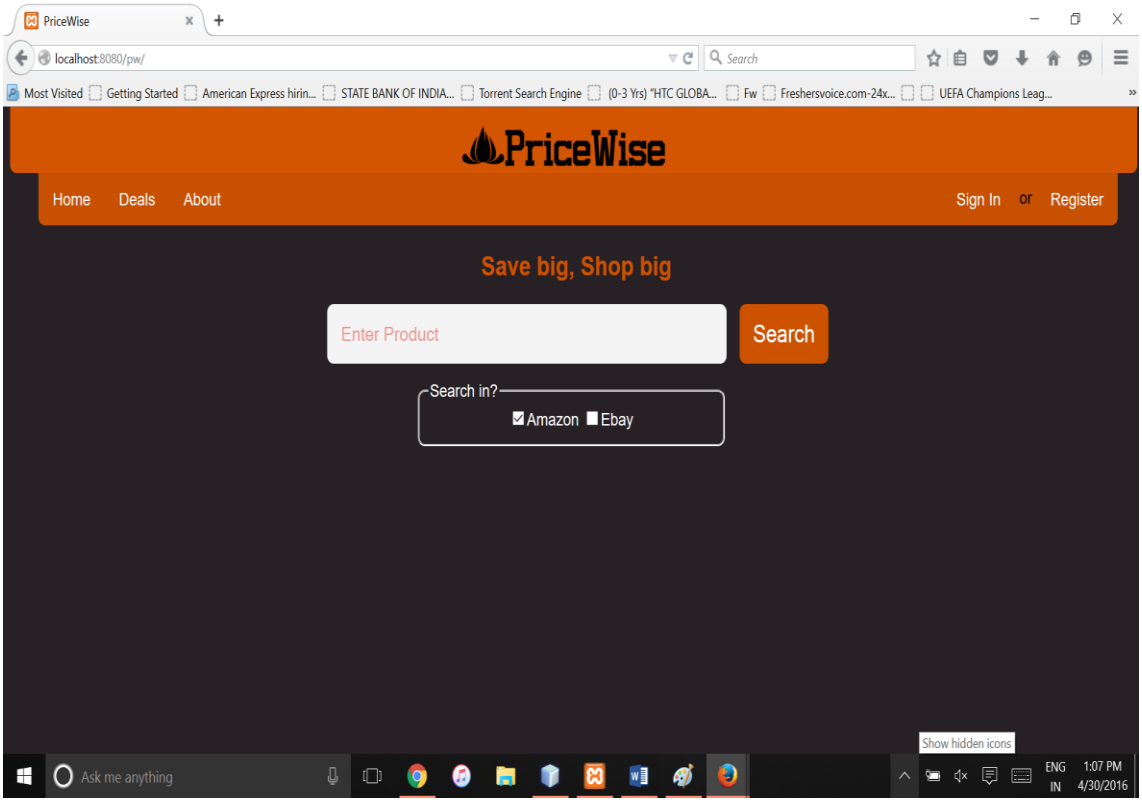
Non Functional Testing and Results:

For non-functional testing, we require the whole system to be tested together. We will be performing the either or all of following non-functional testing.

1. **Configuration Testing**: To ensure the configuration of the system we will be making sure that our application can be used on all browsers.



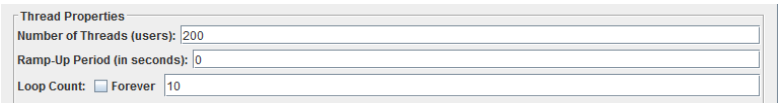
PriceWise on Internet Explorer web browser



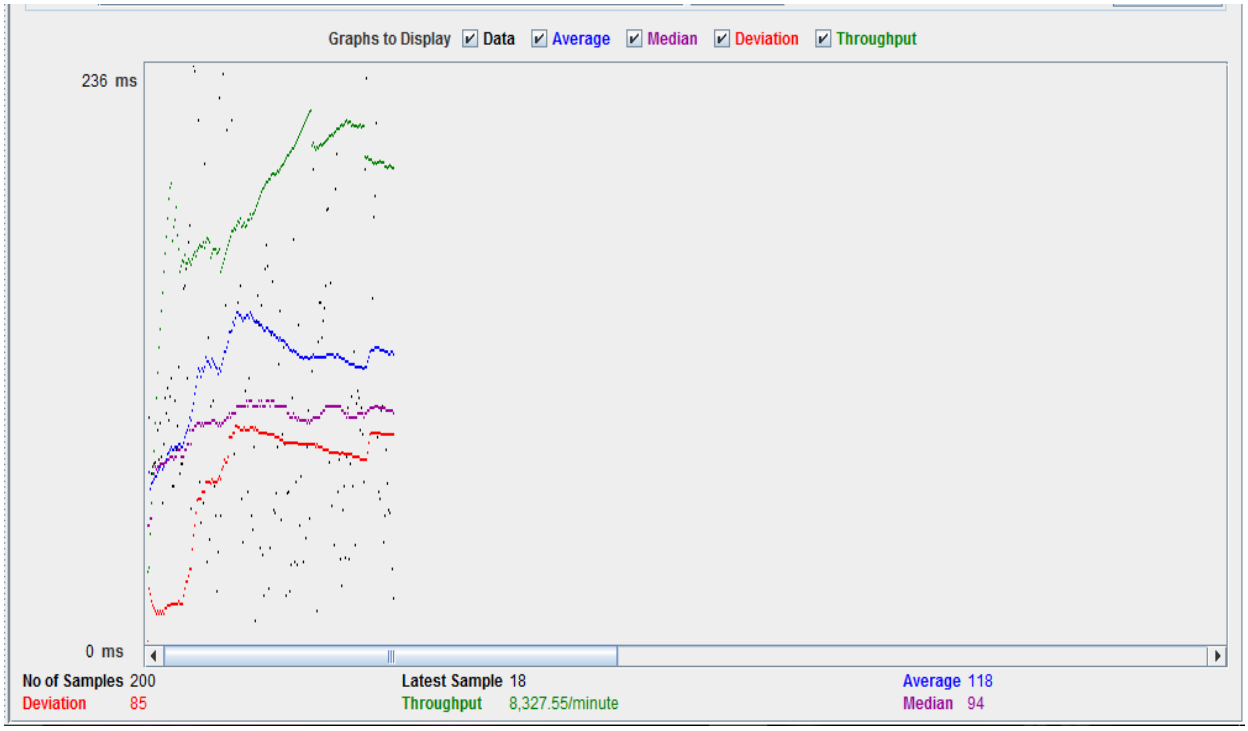
PriceWise on Mozilla web browser

2. **Usability Testing**: Website should be easy to use. Instructions should be provided clearly. We will check if the provided instructions are correct means whether they satisfy purpose. Main menu should be provided on each page. It should be consistent throughout the system. So to make sure our web-app follows the above stated requirements, we asked 5 different non-technical users to use our website and rate it accordingly. **Result is 4/5 stars.**

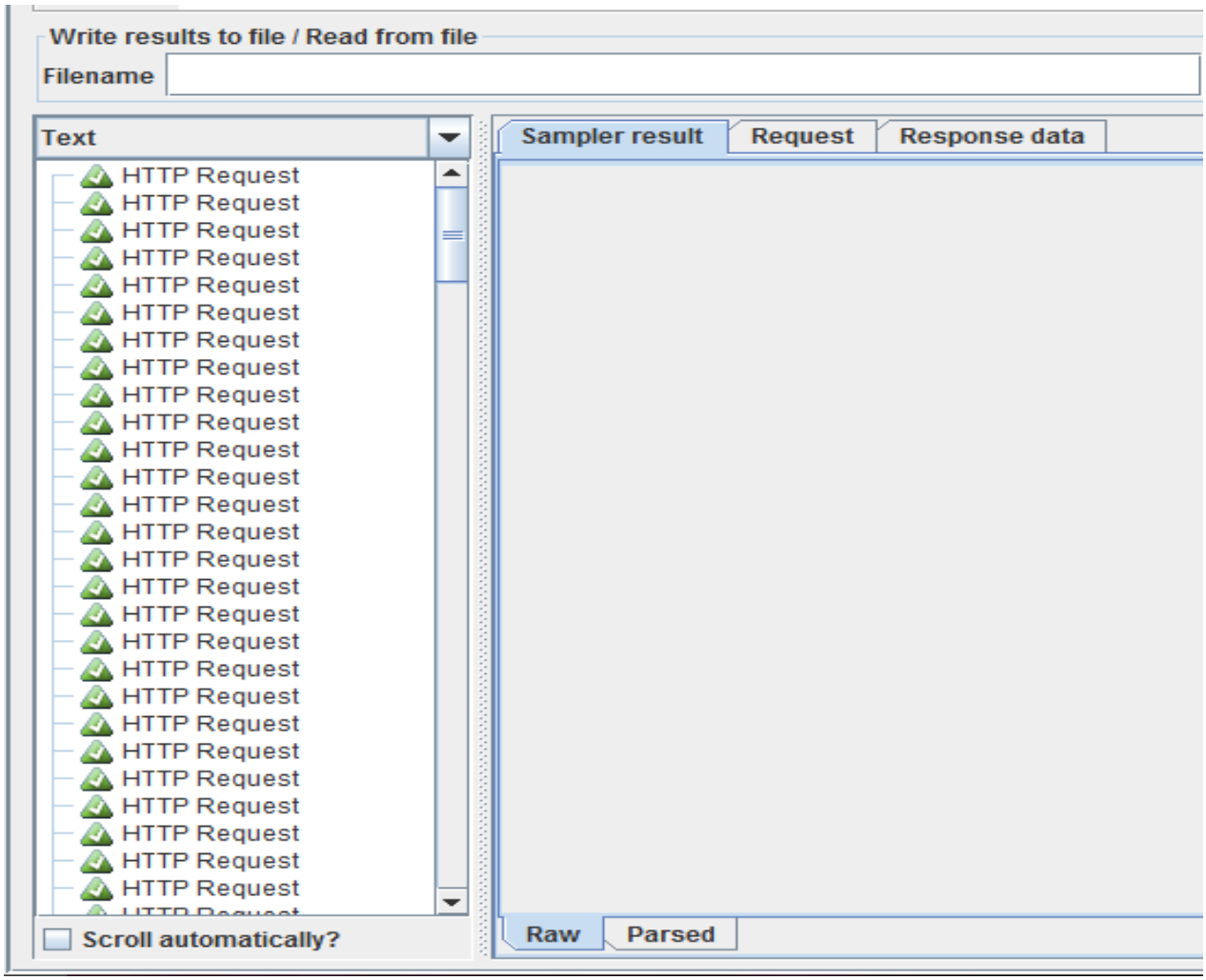
3. **Performance /Load Testing**: Web application should sustain to heavy load. Also we will check our application on the different internet speed to check the performance. We will use JMeter to perform the performance, stress and load testing.
   Load Testing : done with 200 users



Throughput is near 8000/min



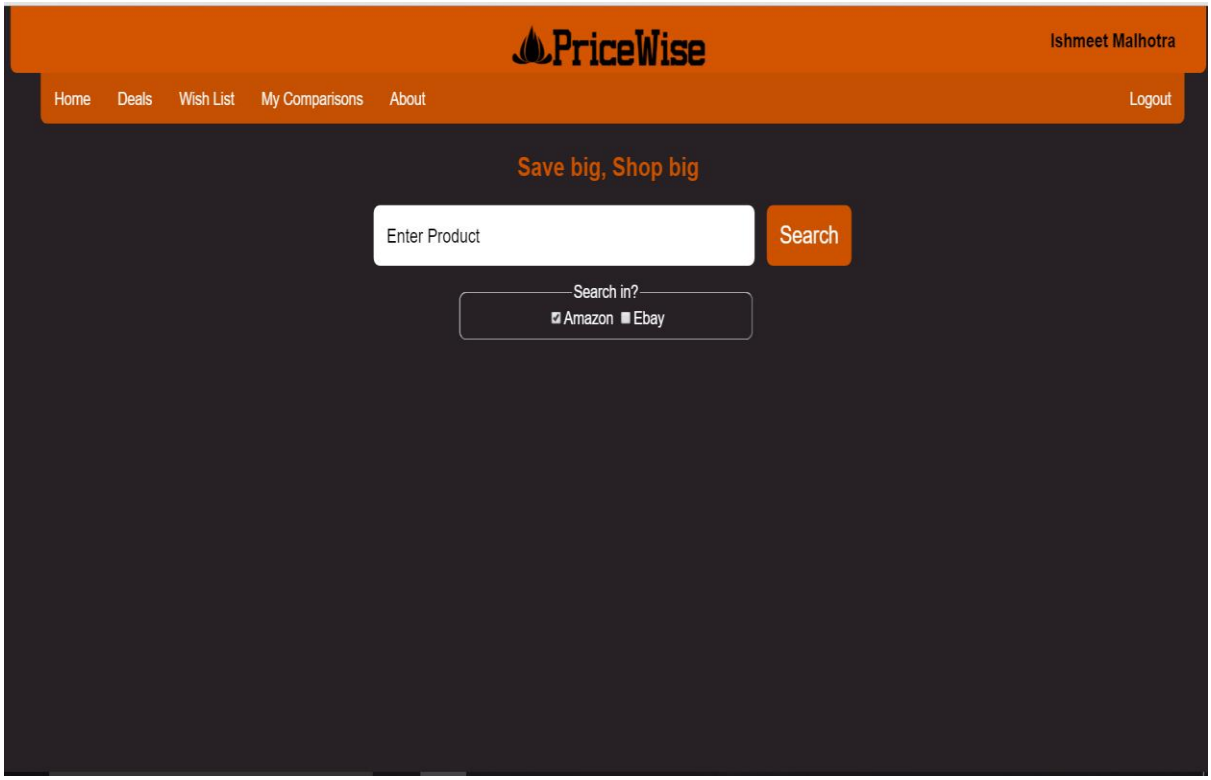
No error encountered



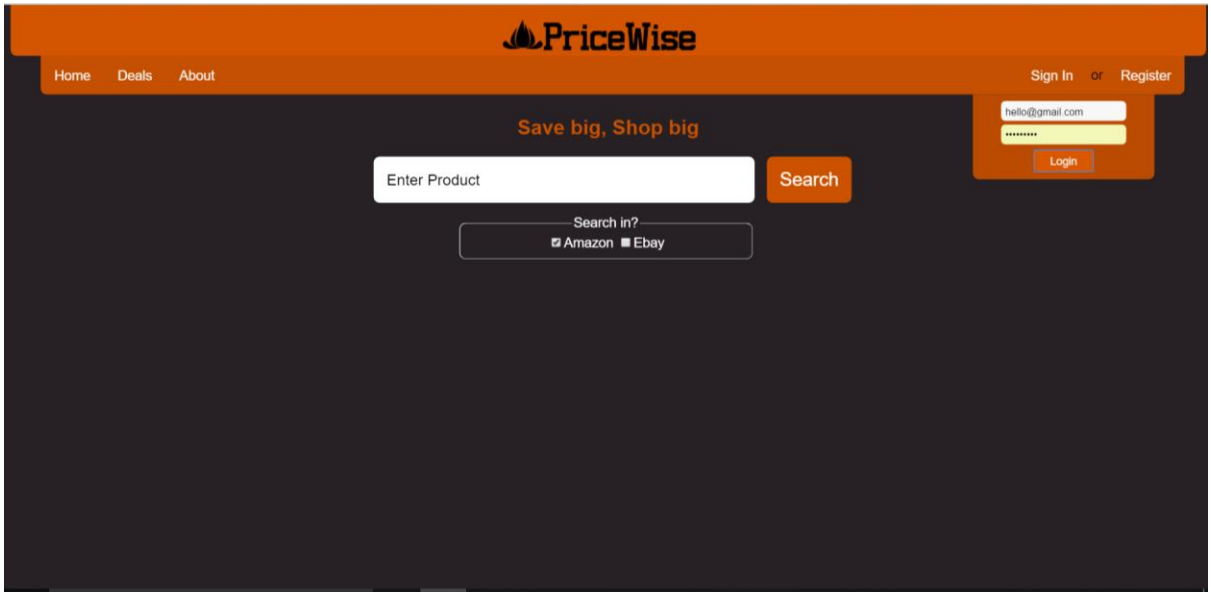
4. **Security Testing**: We will be using Websecurify to perform the security testing. Websecurify is a very easy-to-use and open source tool which automatically identifies web application vulnerabilities by using advanced discovery and fuzzing technologies. Some of the other security features testing are listed in the table below.

Security testing						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Valid user should be able to login successfully into the application.	1. Open the PriceWise application. 2. Enter valid user name and password	User should be able to login successfully and see the user home page.	Test case 01	P	
02.	Verify if user enters wrong user name or password login page should show error message**.**	1. Open the PriceWise application homepage. 2. Enter any invalid user information.	The user is not able to see the user home page and email will keep flashing.	Test case 02		

Test case 01:



Test case 02:



Features to be tested(Functional):

The features described can be mapped to one or more functional requirements and non-functional requirements. We will be using a very simple risk-measuring scale, consisting of three levels of risk for each of the features we are planning to implement and test for the first release of the product. High risk features involves the highest priority features that has to work in order to other system work and are more prone to failure than lower risk features. Medium risk features medium priority features whose working may have little impact on other features. Low risk features less complex and do not have dependencies on other components.

1. **Product selection from different merchant websites**. Risk is high: This feature has dependency on multiple APIs from where the data is been collected and displayed and it is the basic feature of our application for all other features to work. But it is an independent feature. Failure of this feature will result in the failure of the application.

2. **Price comparison of the products selected by the user** Risk – High: This feature involves the product selection feature to work with it. Also it includes multiple API calls so it is a high risk feature.

3. **Getting the reviews of the selected products**. Risk – Medium : This feature works with the product selection

4. **Display the merchant deals**: Risk – Low: Getting the merchant coupon is an independent feature.

5. **Display the price trends of the product**. Risk – Medium: This feature depends upon the product selection and Comparison feature.

6. **User must be able to successfully login into the system**: Risk – Low

7. **User must be able to add/delete products into the wish list** Risk – Medium: This feature is dependable on the product selection and user login feature.

8. **User must be able to able to update the account:** Risk – Medium: The user data should be successfully stored in the database.

9. **User must be able to manage the alerts**. Risk – Low. It will be a standalone feature.

Functional Testing Strategy:

A. **Testing Levels:** This describes our general approach at the different levels of testing of our system.

1. **Code inspection:** Team members will follow code style standards to ensure readability of the source code. Apart from this, we will be carrying out code inspections before any contributions are added to the main code-base. Contribution will go through a code inspection process and will need to be approved by the team members. Code inspection will be performed on all components of the system.

2. **Unit testing:** Unit testing will be done individually by developers with the help of Code Igniter Unit Testing to perform the automated testing. The test cases and outcomes will be discussed with the team. Test cases picked for unit testing will follow *equivalence partitioning strategies black-box technique* to test for correct behavior on general

testing domain inputs and boundary cases. The metrics to be collected at this testing level are total number of test cases, number of test cases passed, number of test cases failed and test coverage. We will do the unit testing manually.

Equivalence partitioning strategies black-box technique: In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

3. **Integration Testing**: Our team will make use of incremental testing using a bottom-up approach to test components after these are developed and their dependencies are unit-tested. Employing a bottom-up approach will also guide our development process to be focused on individual, higher risk components first and inter-component communication later. For instance in the case of our product , we would first focus on building base components such as product selection and then the components that depend upon this feature like price comparison and getting product reviews. We will perform the integration testing manually. At this stage we will collect metrics about the number of passing and failing integration tests and coverage.

4. **Regression Testing:** As the system grows and more features are added we run the risk of having some features conflict with each other. To handle this, the team will run smoke tests before merging any new features to the main code base. These tests consist of a subset of unit and integration tests.

5. **User Acceptance Testing:** The acceptance of our product will be determined by the product owner. The product owner will determine whether or not the pass/failed criteria for each of the features our requirements need to support are met. This stage of testing will be done after all integration tests are met. The product owner will walk through a

set of system use cases that make use of the supported features and check for completeness and support of the system. This will cover the entire system as a whole.

B. **Approach for planned test Cases**: We will follow the equivalence partitioning strategies black-box technique to prepare the test cases.

C. **Bug tracking:** For the issue or bug tracking we will be using Trello.

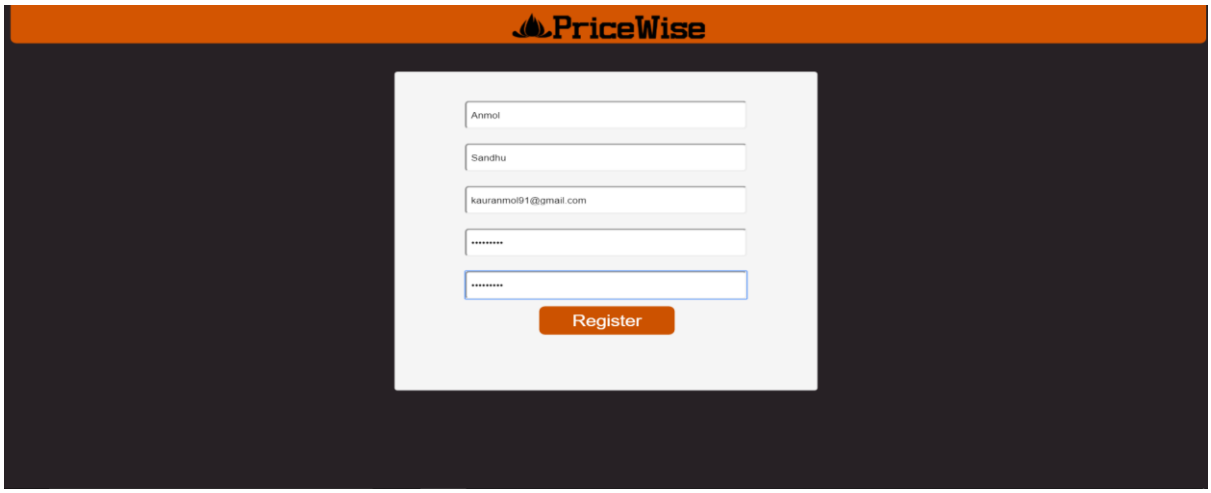
The goal of our test plan is to carry out testing during all our product development. We will incrementally integrate new features as these are implemented. However we will limit our testing efforts to avoid wasting resources. At unit level, testing will stop if the sub component being evaluated passes all proposed test cases by the developer and the team members. At integration level, testing will stop if the proposed test cases by the team pass and the feature to be integrated with the main code base doesn't interfere with the existing features in the code-base at that point in time. Testing in general will also stop if we encounter technical difficulties with the platforms and frameworks being used and would be resumed as soon as these problems are solved. All our testing efforts will finish when we meet the pass/fail criteria for the project and the features.

Test cases and Results

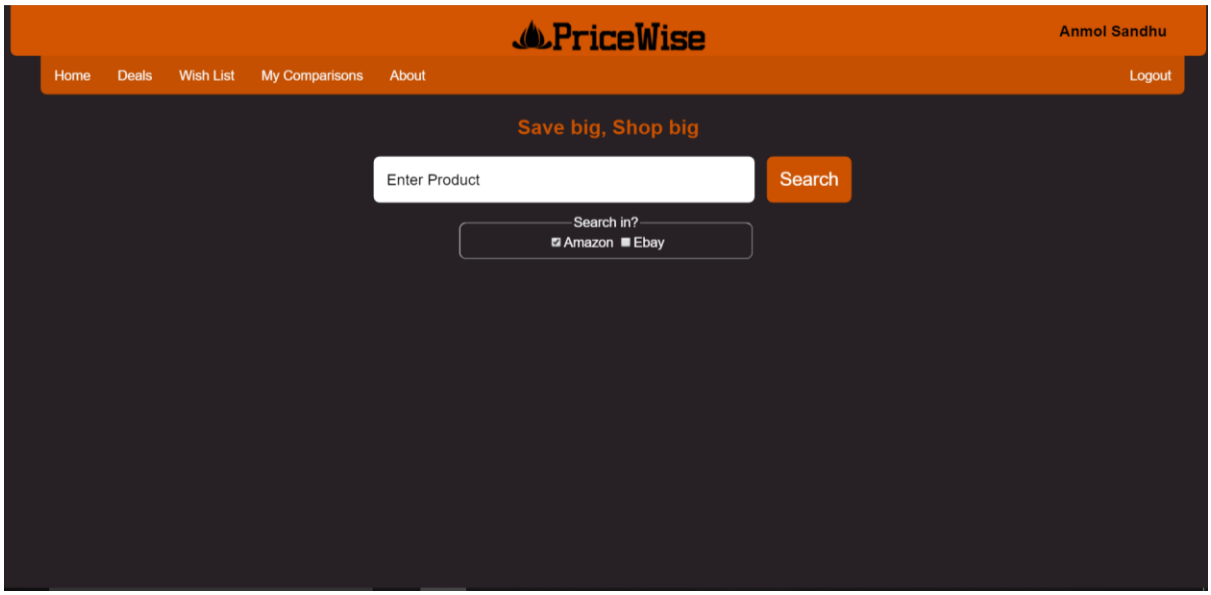
We have provided some of the test case plans with their priority of testing.

Login Module High Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Valid user should be able to login successfully into the application.	3. Open the PriceWise application. 4. Enter valid user name and password	User should be able to login successfully and see the user home page.	Test Case 01		
02.	Verify if user enters wrong user name or password login page should show error message.	3. Open the PriceWise application homepage. 4. Enter any invalid user information.	The user is not able to see the user home page and email will keep flashing.	Test Case 02		
03.	Verify if new user is able to sign up to create new account.	1. Open the PriceWise application homepage. 2. Enter last name, first name, email, username, and password.	The user must be able to create new account. The new user is stored in the User Table in the database.	Test Case 03		
04.	Verify if all the validation checks have been successfully implemented while creating the new user account.	1. Open the PriceWise application homepage. 2. Enter invalid last name, first name, email, username, and password.	The error message should be successfully displayed. User information should not be stored in the user table			

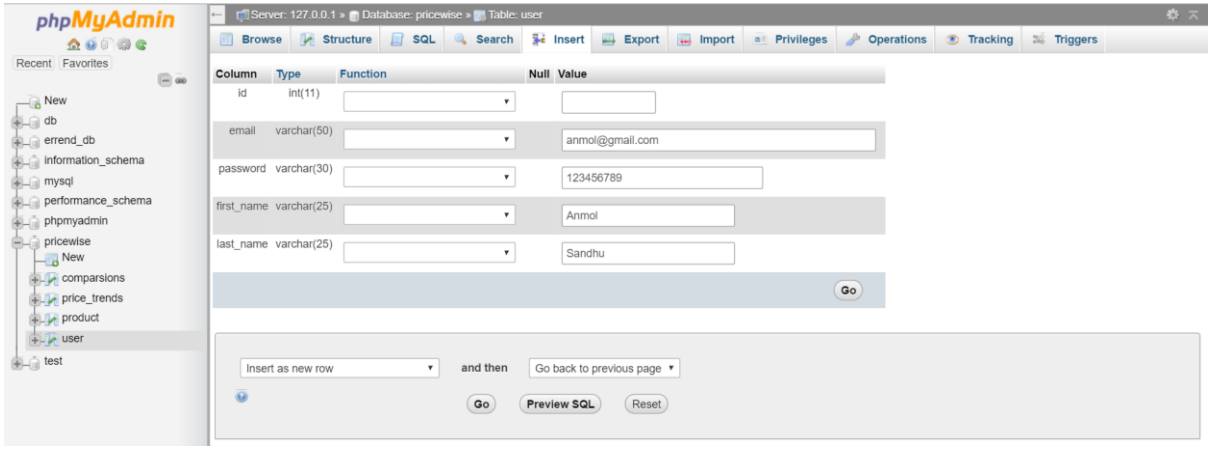
Test case 03:



New User is able to login successfully



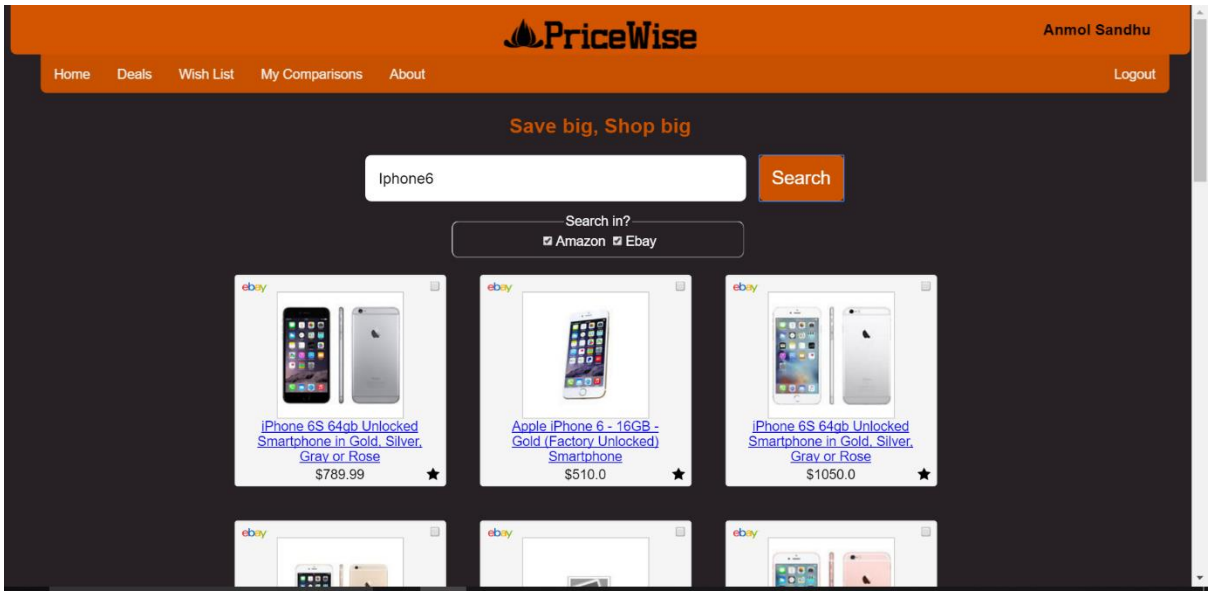
Database:



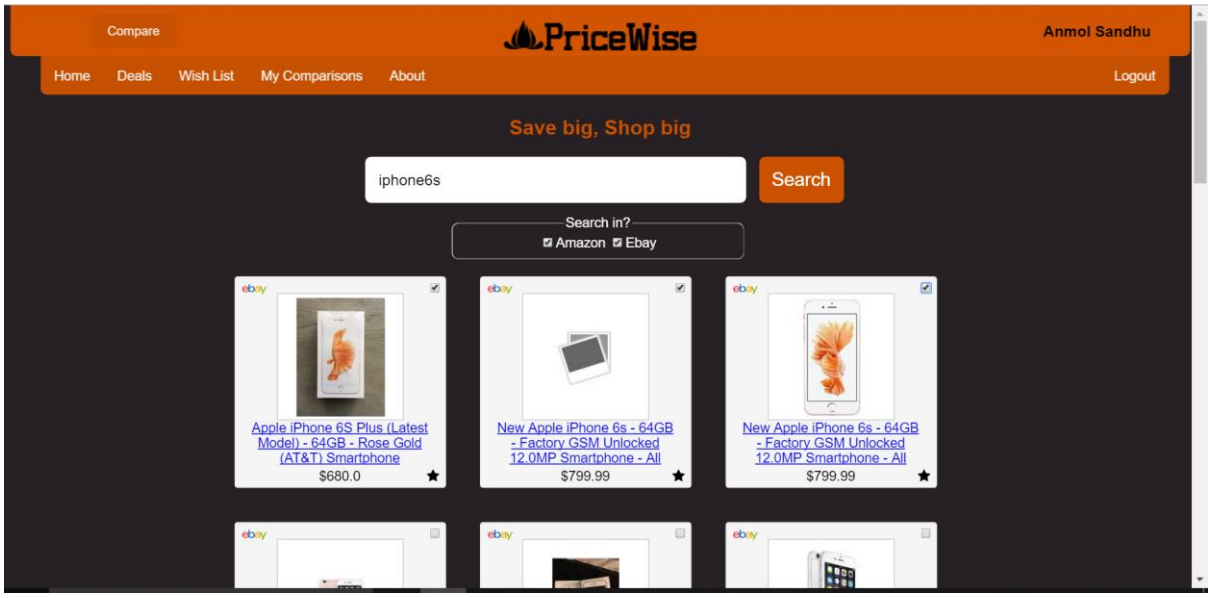
Product Selection and Comparison Module - High Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Verify if the user is able to search the product for price comparison	1. Open the PriceWise application. 2. Select the merchant website for the product selection for the dropdown menu. 3. Enter valid keyword for the product in the	Product list should be displayed successfully	Test Result 01		

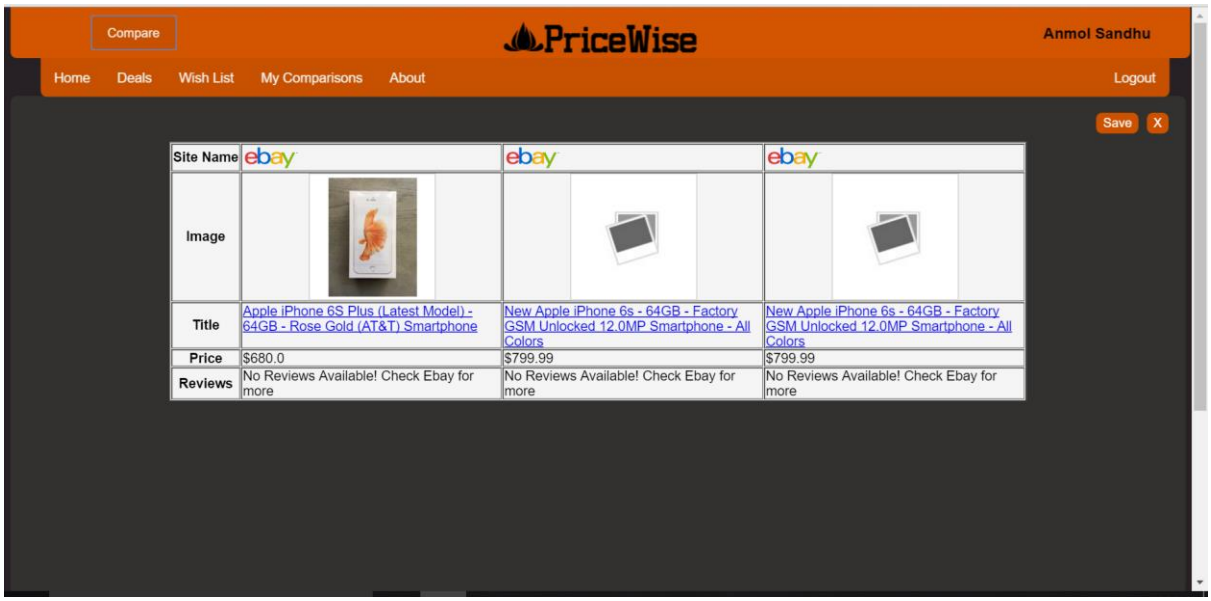
		search tab and press enter				
02.	Verify if the user is able to select the products to compare the price	1. Open the PriceWise application homepage. 2. Click on the check boxes provided on the searched products.	User must be able to select the products to compare the price.	Test Result 02		
03.	Verify if the price comparison is done successfully	1. Select the products to be compared 2. Press enter to compare price	The user must be able see the price comparison of the selected products successfully.	Test Result 03		

Test Result 01:



Test Result 02 – 03:



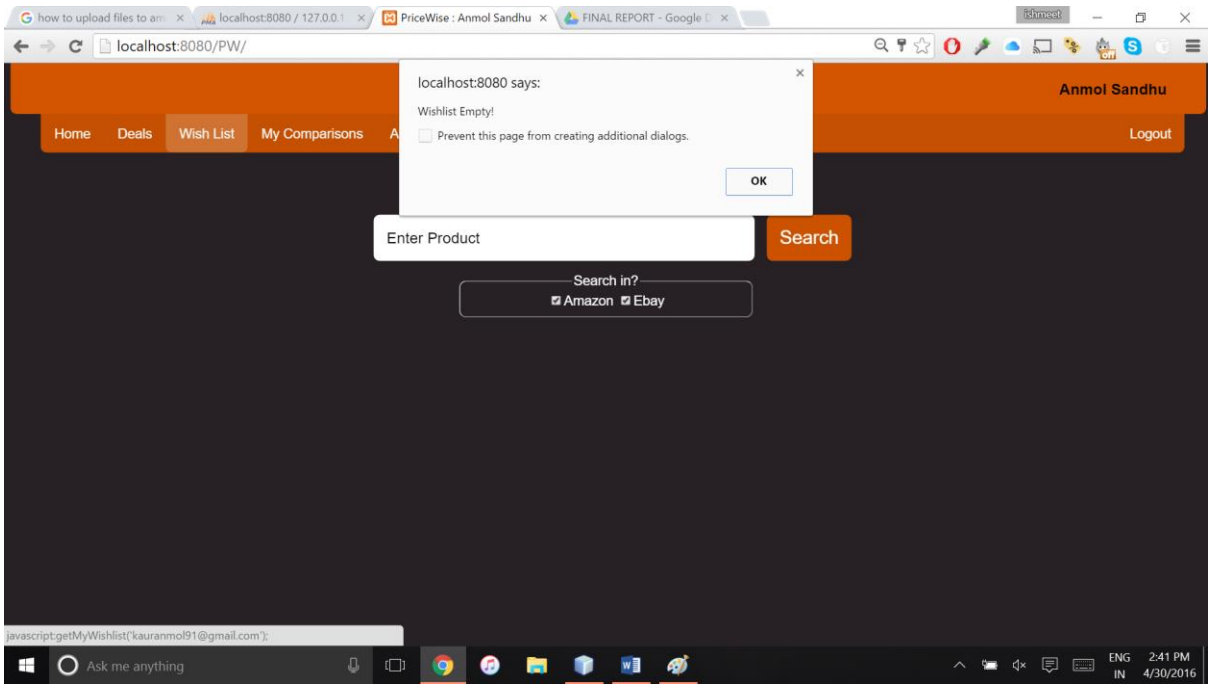


Wishlist Module High Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Verify if the user is able to add product to the wishlish.	1. Login into the user account. 2. Search a product and press key “add to wishlist”	User should be able to login successfully. Product should be added to wishlist tab and in the wishlist table in the	Test Result 01		

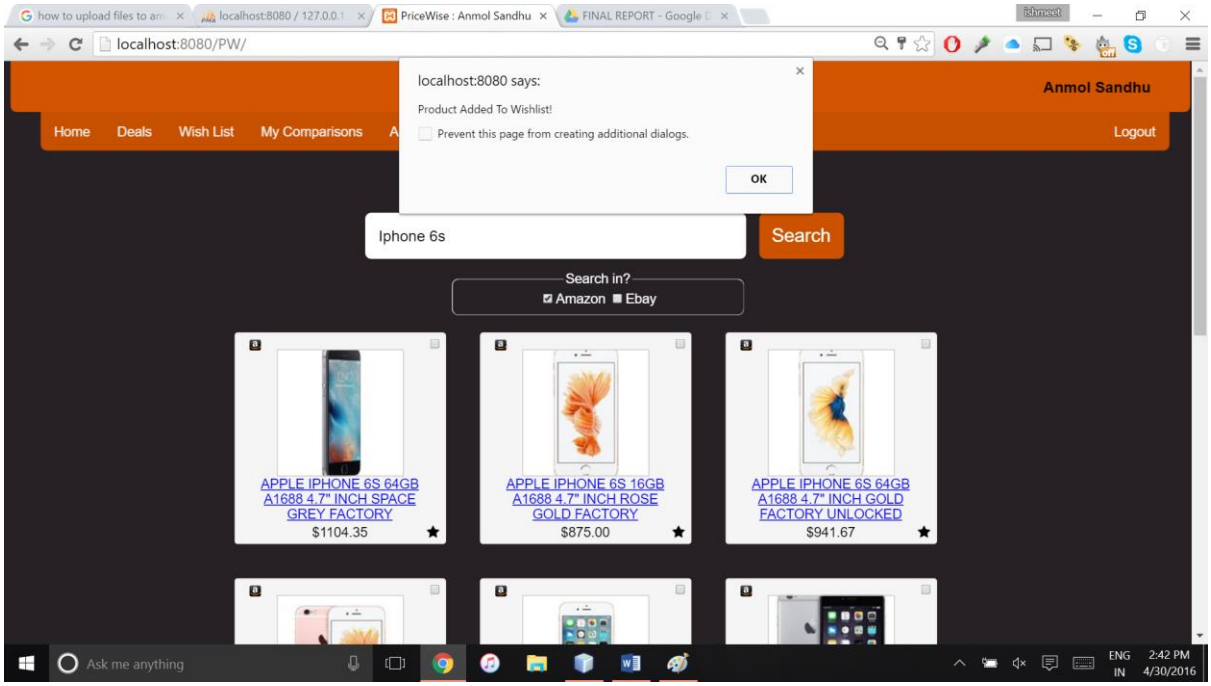
			database.			
02.	Verify if user is able to remove the product from the wishlish	1. Login into the user account. 2. Open the user wishlist 3. Press key to remove the product from the wishlist	The product should be removed from the user wishlist and wishlist table in the database.	Test Result 02		
		4.				

Test Result 01:

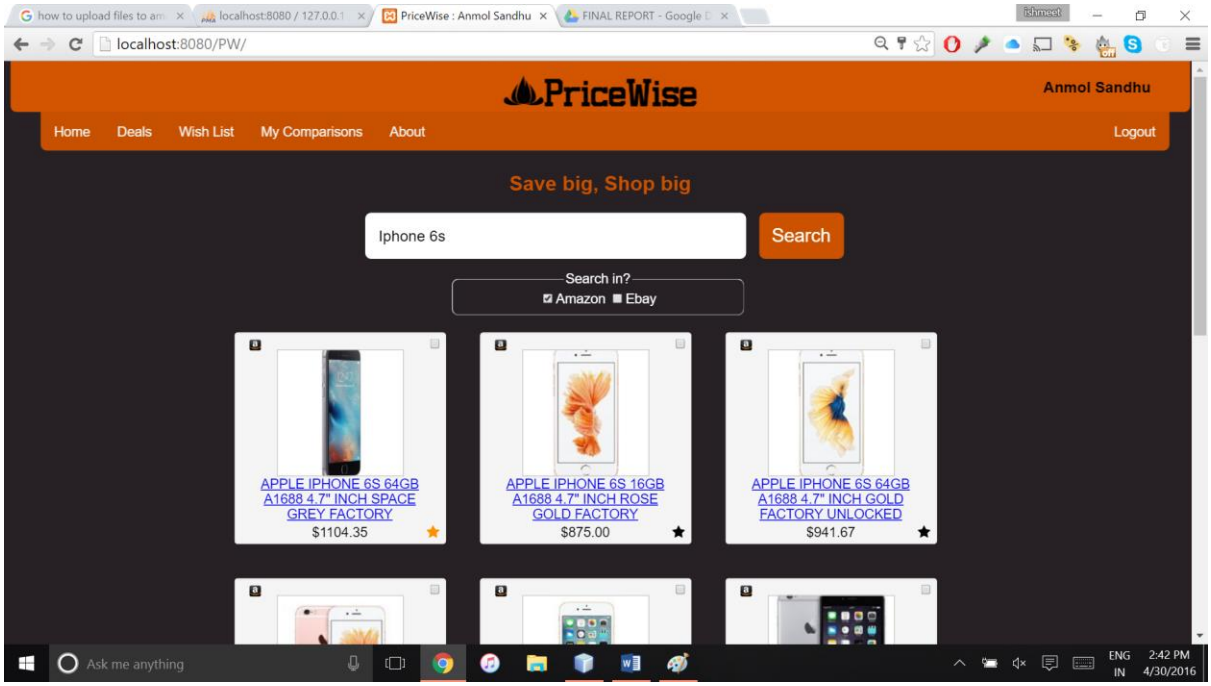
When Wishlist is empty:



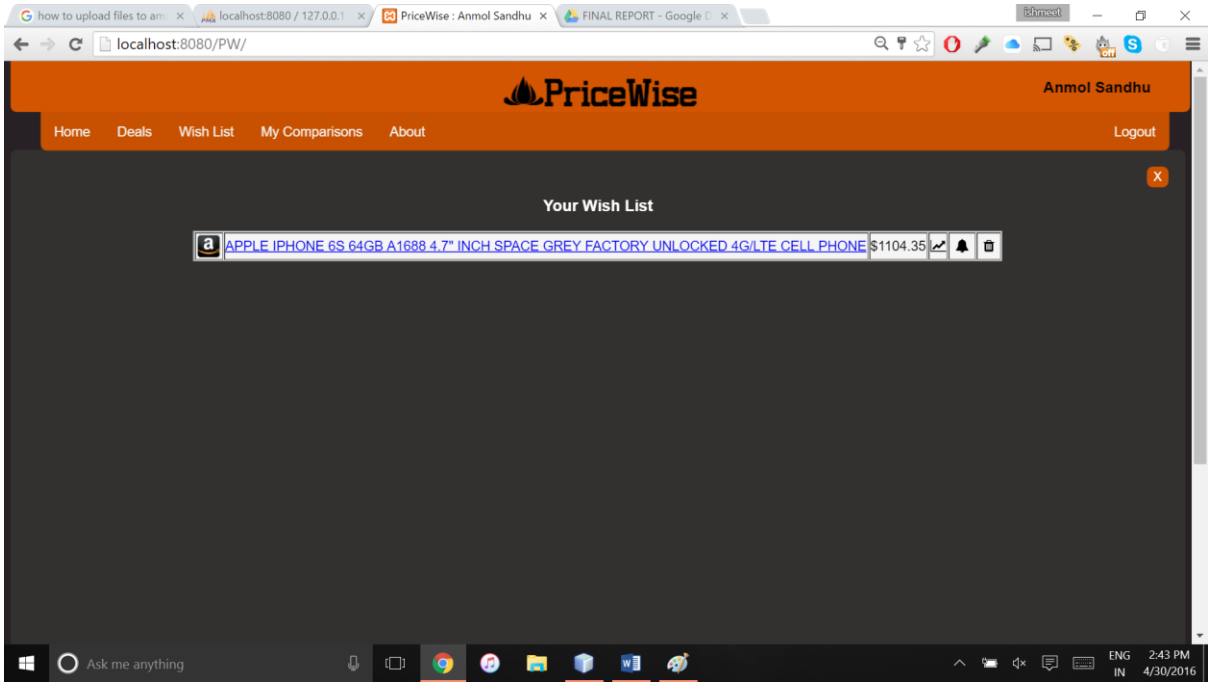
When we add product to the wishlist



Yellow star shows that product is added to the wishlist

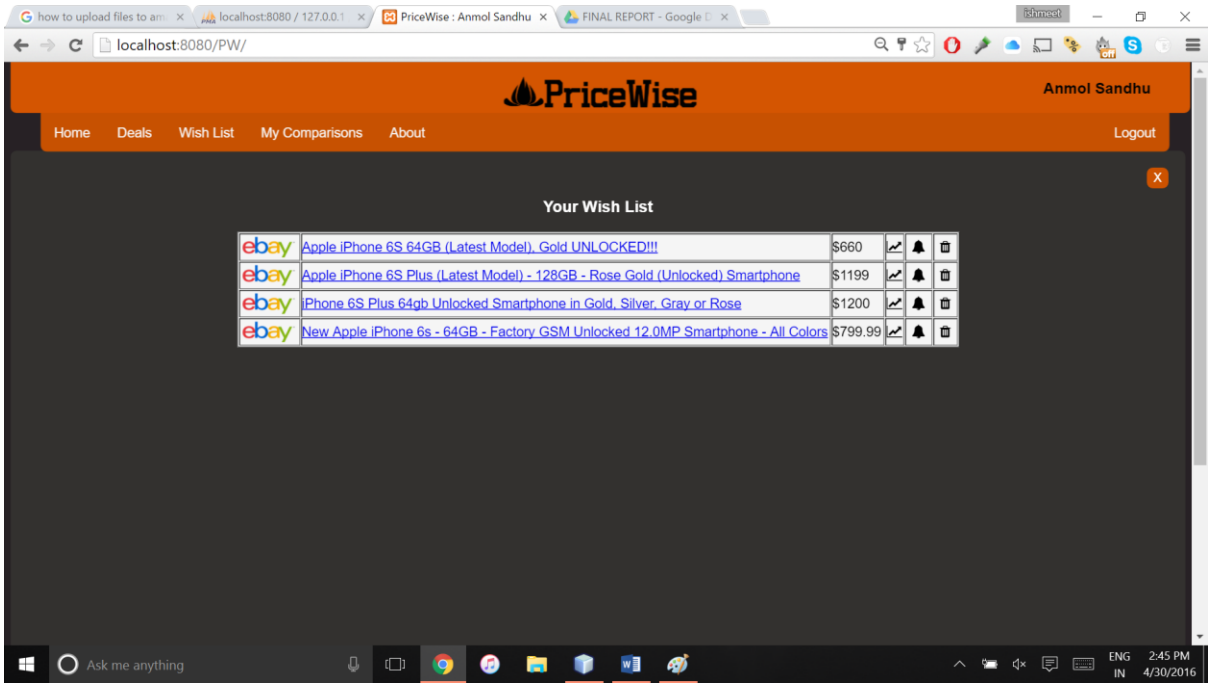


User Wishlist:

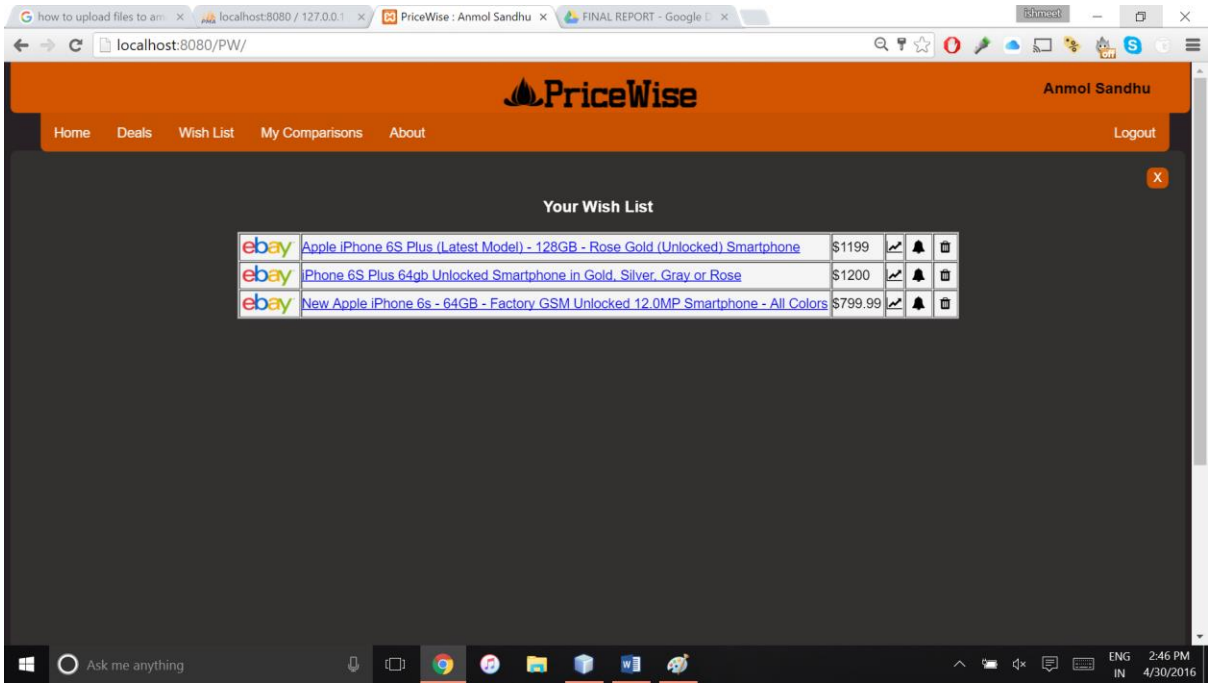


Test Result 02:

User has currently 4 products in the wishlist

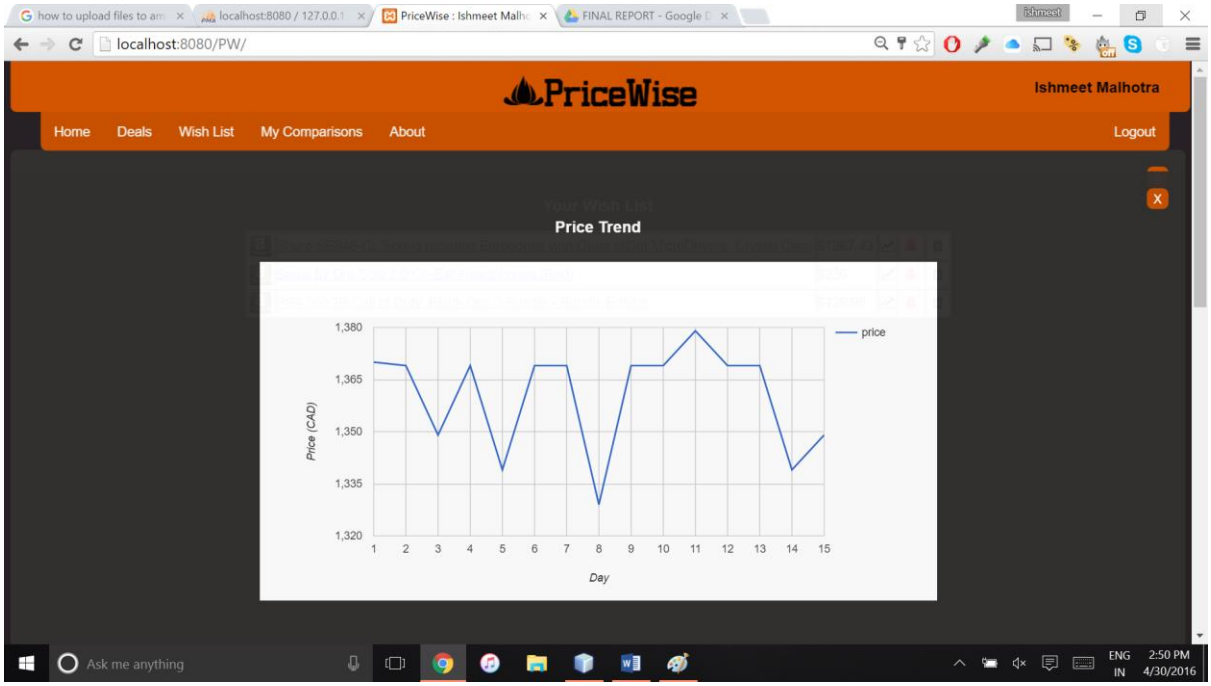


If we remove one product:



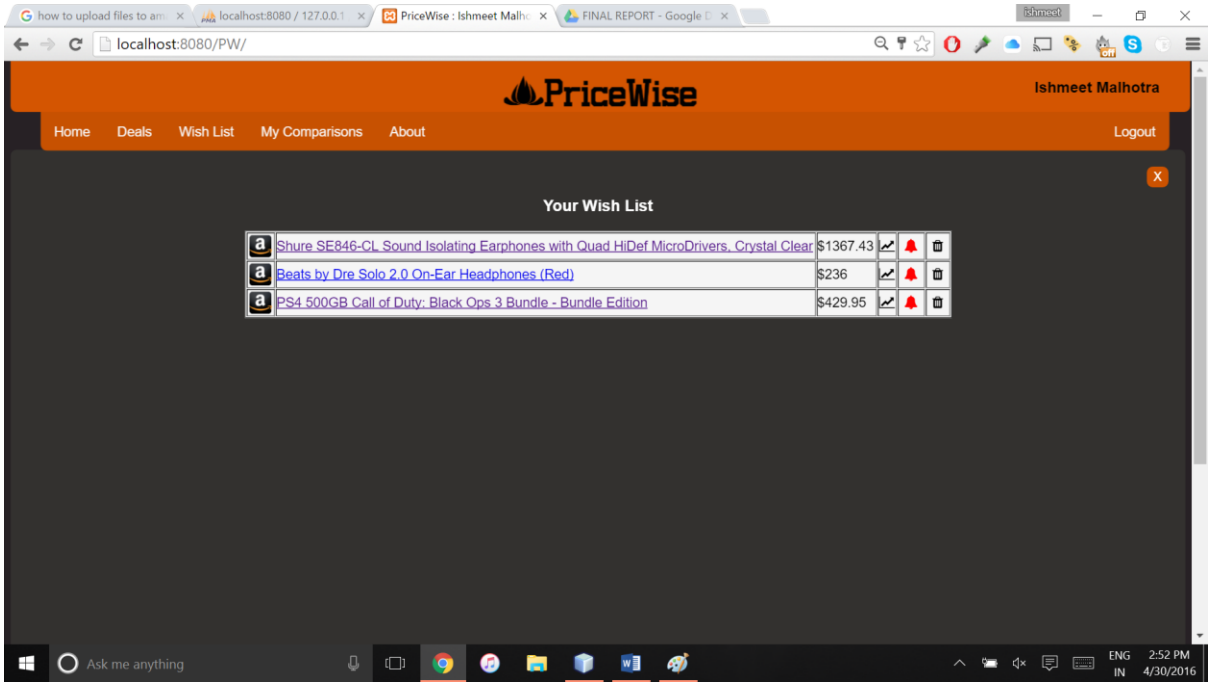
Price Trends: High Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Verify if the user is able to retrieve the price trend of the product	1. User must be logged in the system. 2. Display on the product price trend to get the price graph.	Price trend graph should be displayed.	Test Result 01		

Test Result 01:

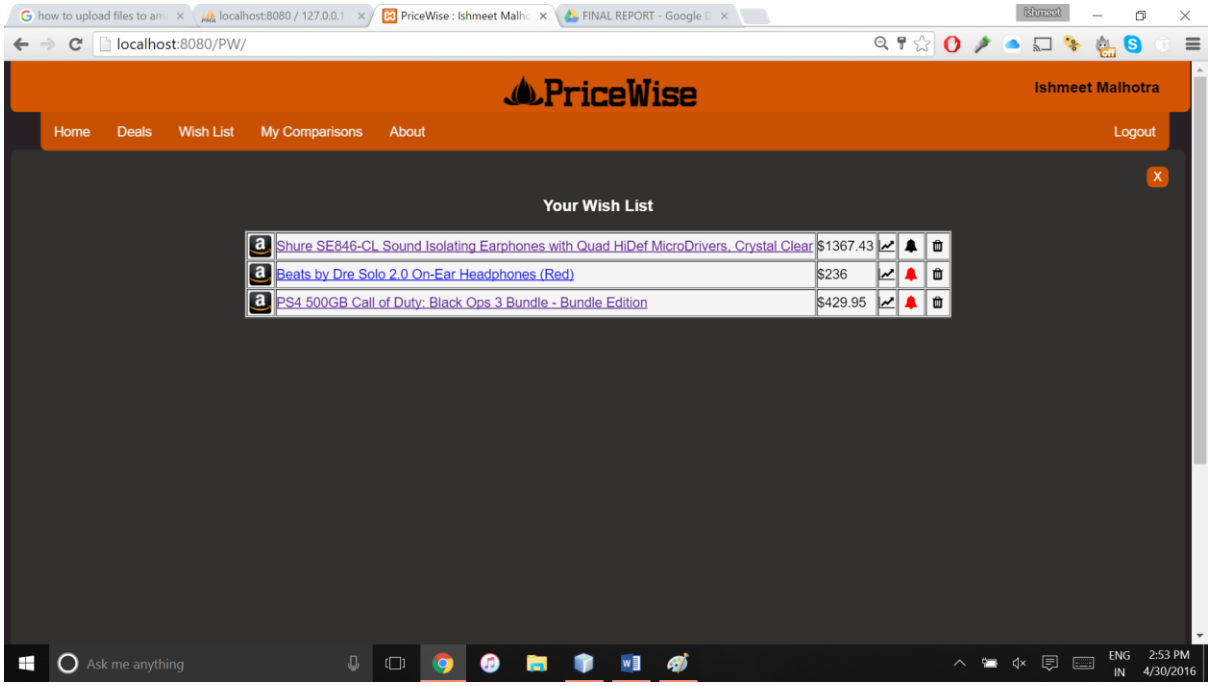


Manage Alert Module: Medium Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Verify if the user is able to set the alert for the product price	1. User must be logged into the system. 2. Select product and press key " set alerts"	User should get alerts for the price change for the selected product.	Test Result 01		
02.	Verify if user can change the alert preferences	1. User must be logged into the system 2. Goto the alerts tab and set the preferences to weekly or daily alerts.	User must be able to change the alert settings	Test Result 02		

Test Result 01-02:



The red color shows that the alert has been enabled.

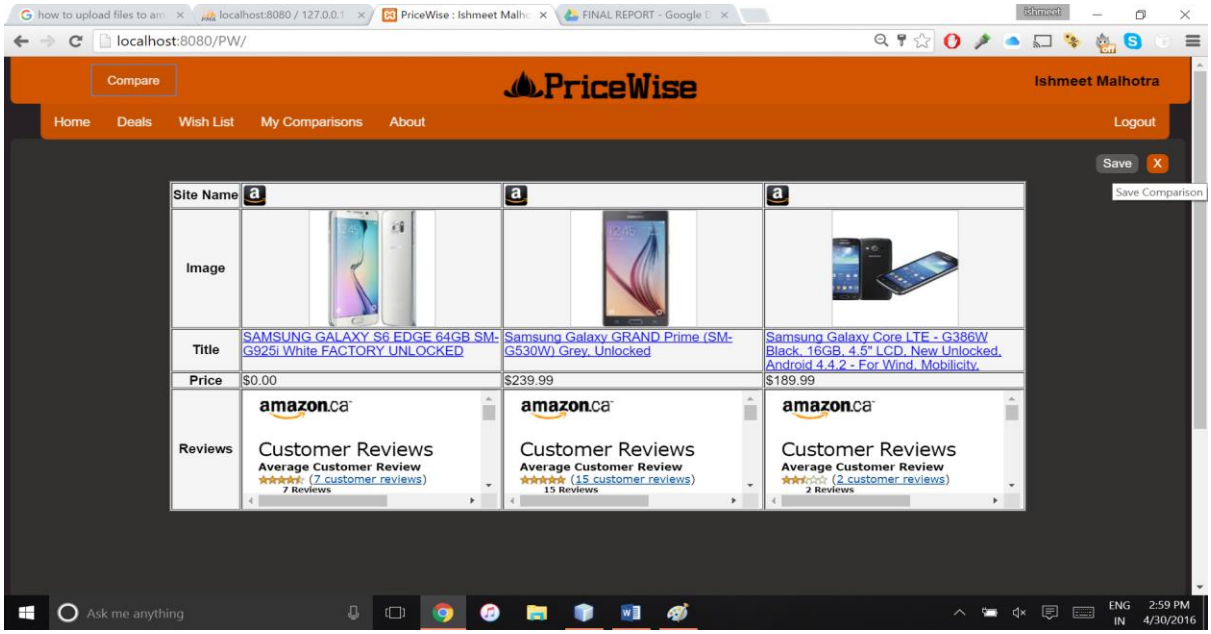


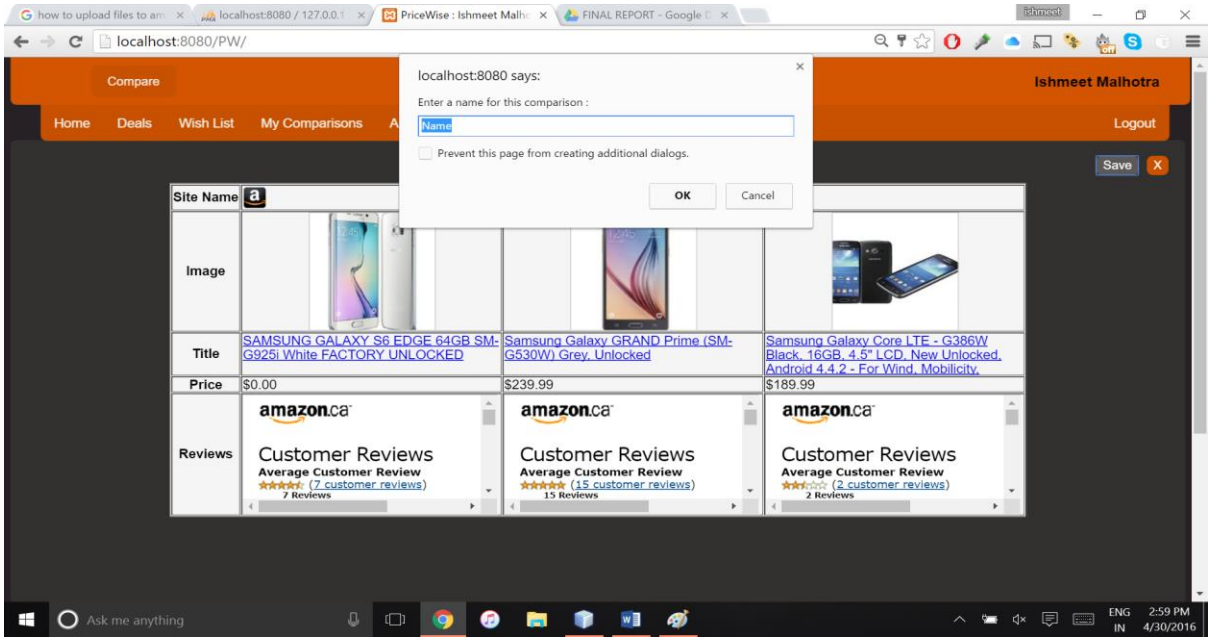
After we disable the alert.

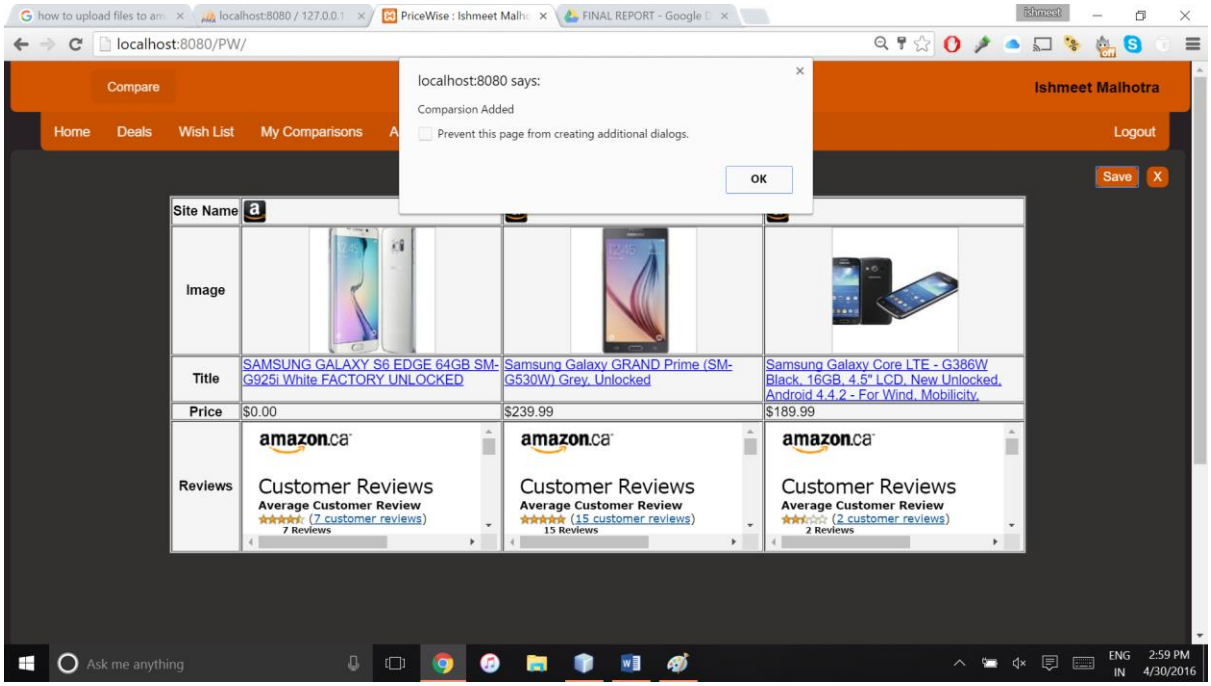
Save Comparison: Medium Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes

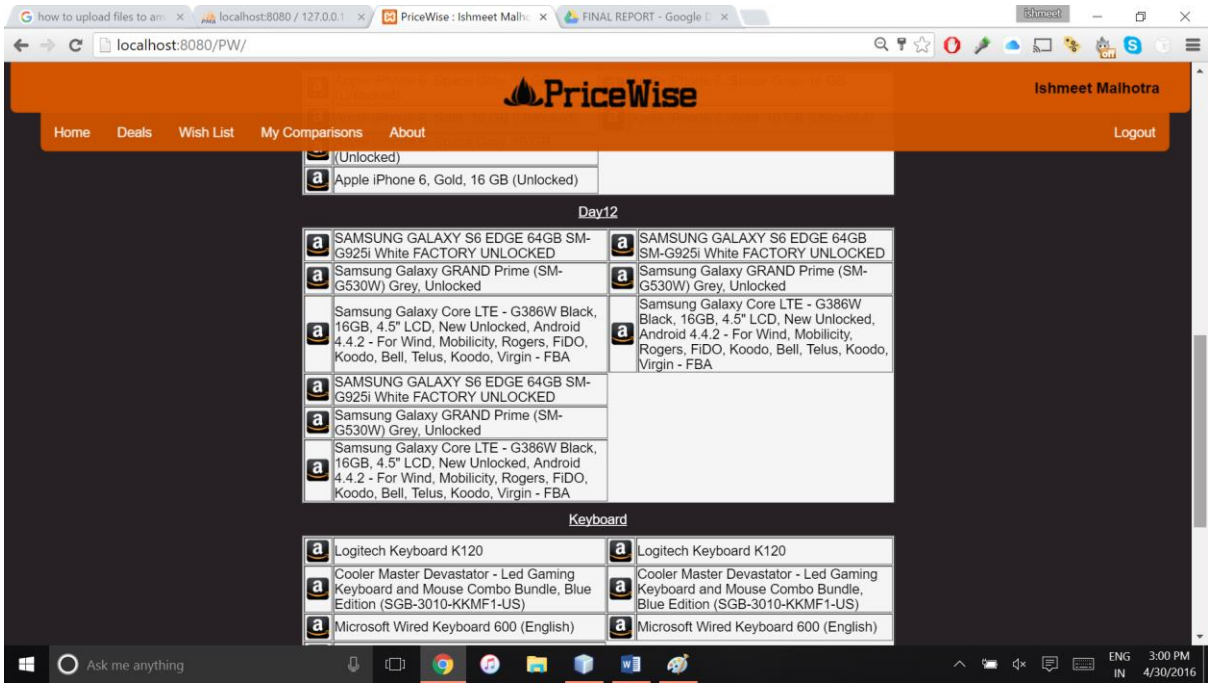
01.	Verify if the user is able to save the comparison	2) User must be logged into the system. 3) Compare the products and save the comparison history.	User should save the comparison.	Test Result 01		

Test Result 01:





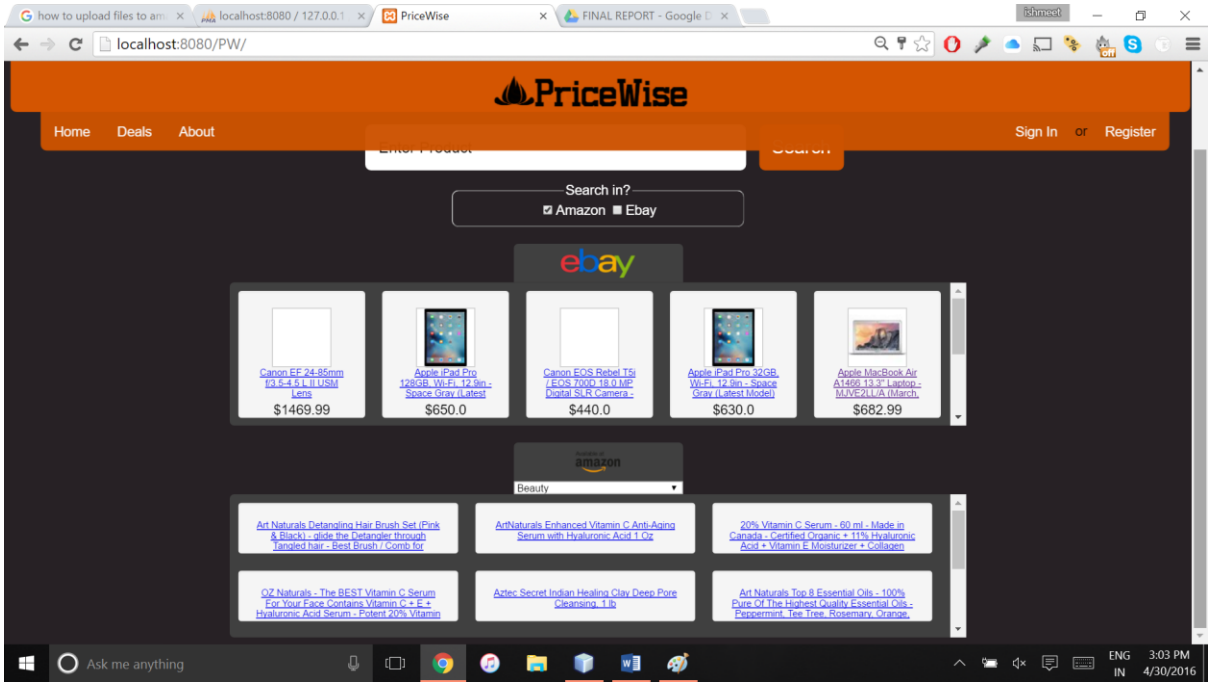




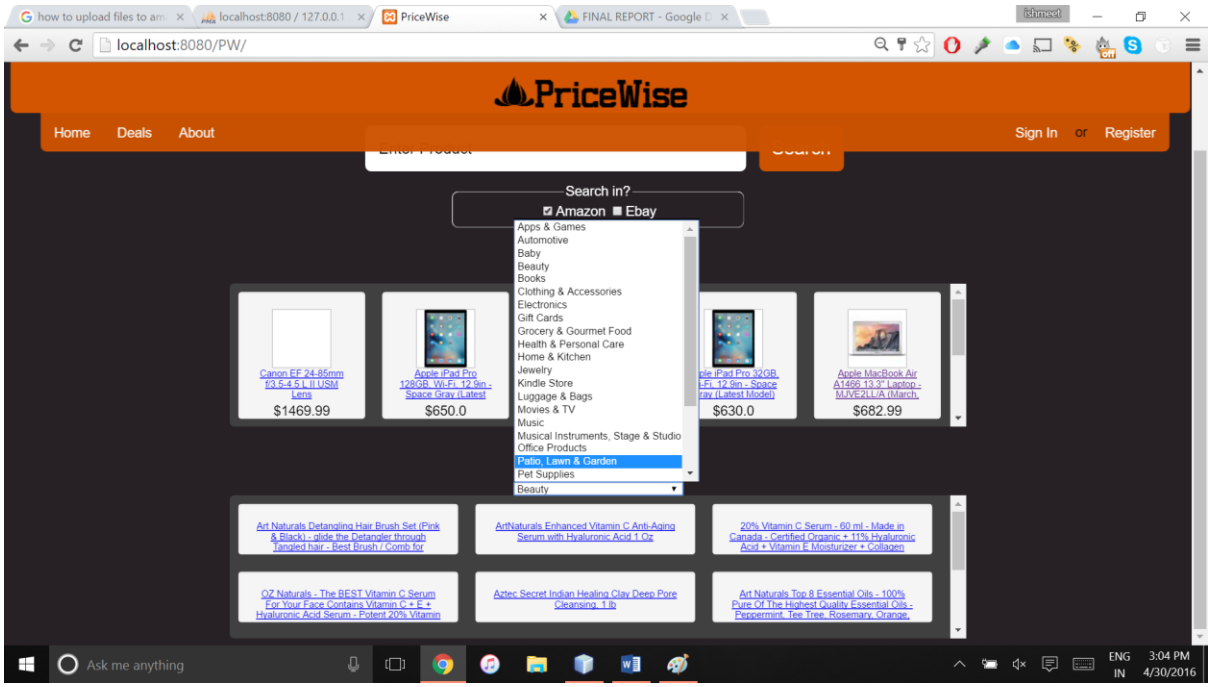
Daily Deals: Medium Priority						
Test #	Requirement Purpose	Action / Input	EXPECTED RESULT	Actual Result	P/F	Notes
01.	Verify if the user is able to see the daily deals from the specified merchant	The user must click on “Deals” Button	Daily Deals should be displayed	Test Result 01		

Test Result 01:

For Ebay:



For Amazon:



Risks and Contingencies

a) **Difficulties understanding testing framework:** As we are using multiple technologies and testing frameworks, encountering difficulties with operating these will result in delays in testing.

b) **Lack of time to complete all test tasks:** We may find ourselves short of time to complete all test tasks specified in this plan. In this case we will prioritize the tasks that involve high risk features.

c) **Team member unavailability:** The testing plan requires all team members to actively be involved in testing. If the team member who implemented the feature to be tested is unavailable, other team member will test the feature.

User Manual:

Pre-requisites:

- A local server installed on your system. Try XAMPP for windows and MAMP for Mac OS.
- A web browser

Please follow the steps below to successfully install web-app on your system:

WINDOWS:

- Place PW folder in the htdocs folder inside XAMPP.
  Your path will look something like: **C:\xampp\htdocs\pw**
- Make sure your Apache is running on port 8080
- Go to phpmyadmin
  Your parth will look something like: localhost:8080/phpmyadmin/
- Create a new database and name it pricewise
- Import the provided sql file into it
- Use this link to add the php script to update the prices and send alerts as a scheduler task https://www.youtube.com/watch?v=sx4vh4KdFPw
  PHP controller name is UpdateProductsController.php
- Go to localhost:8080/pw/ from your browser to run the project.