Requirement Specification

Document

***Project OnShop***

Stephen Knofczynski, Brian Hawkins, Andrew Vaughn

Team OnShop, University of Nebraska at Omaha

March 29th, 2018

Abstract:

The purpose of our project is to create an online shop where users can create an account, and purchase items. The website contains four basic features, Products, Users/Accounts, Feedback, and Support. Products is a section of the website where a user can search for products in the database. This will include features such as items related to the specific one users are searching for, as well as popular or items that are trending. Users/Accounts will contain various personalization features, such as products they love, what is in their shopping cart, and their personal info/bio they can edit while logged in. Feedback will be available for every product. When someone purchases an item, they are allowed to leave feedback, and give the item a rating 1-5 based on their experience in the purchase of the item, and the use of the item. Support will be given to users that will be a live chat feature so that someone can be connected with a representative of the store in case the user experiences a problem with the site.

**Table of Contents**

Table of Contents

**Problem Statement and Project Proposal1**

**Use Cases and Functional RequirementsUse Cases and Functional Requirements3**

**2.1 Request: index.html**

5

**2.1.1 Request: index.html documentation**5

**2.1.2 Request: index.html Functional Requirements**5

**2.2 Request: register.html**

**2.2.1 Request: register.html documentation**

**2.2.2 Request: register.html Functional Requirements**5

**2.3 Request: login.html**

**2.3.1 Request: login.html documentation**

**2.3.2 Request: login.html functional requirements**6

**2.4 User submits registration**

**2.4.1 User submits registration documentation**

**2.4.2 User submits registration functional requirements**6

**2.5 User submits login**

**2.5.1 User submits login documentation**

**2.5.2 User submits login functional requirements**7

**2.6 Request: profile.html**

**2.6.1 Request: profile.html documentation**7

**2.6.2 Request: profile.html functional requirements**

**2.7** **Send data**

**2.7.1 Send data documentation**

**2.7.2 Send data functional requirements**

**2.8** **Receive data**

**2.8.1 Receive data documentation**

**2.8.2 Send data functional requirements**

**2.9 Item added to checkout**

**2.9.1 Item added to checkout documentation**

**2.9.2 Item added to checkout functional requirements**

**2.10 User updates profile**

**2.10.1 User updates profile documentation**

**2.10.2 User updates profile functional requirements**

**2.11 Add product to Database**

**2.11.1 Add product to Database documentation**

**2.11.2 Add product functional requirements**

**2.12 Admin Deletes Profile**

**2.12.1 Admin Deletes Profile documentation**

**2.12.2 Admin Deletes Profile Functional Requirements**

**Non-Functional Requirements**

**References**

**Appendix**

1. **Problem Statement and Project Proposal**

In today’s world online shopping has grown tremendously. Companies such as Amazon have seen more than a 700% increase in stock over the past five years. With all this hype surrounding online shopping most of the big retail companies are doing everything they can to get as many users as possible. However, other than their brand image most online shops are designed the same using a simple left to right shopping experience. Shopping/Website features are on the left, items user looking to purchase are in the middle, and Accounts/Checkout are on the right. This design is done to give the user a sense of progress in their online shopping experience. The simple movement of left to right gives a feeling of start to completion as they proceed to the checkout.

Our online shop intends to set itself apart from the other ones by incorporating a design based more on a GUI and enhanced personalized account customization features. The shop will allow users to drag and drop items they want into a bag for purchase. The site by default will allow users to navigate around it using buttons to give a more GUI feel to it, rather than searching for a feature/product and pressing enter.

In addition to the GUI users will be able to edit the layout of the site to their personal preference. If they do not like the GUI look or feel they can choose to make it into a drop-down menu instead. Users who register an account with the website (required for purchases) will have a personal account page that can be fully customized. This will be done to reflect the surge of social media where users have a personalized page they can edit and control that reflects themselves. For example, users can edit what happens when they drop an item in a bag. They can either keep shopping (default), go to checkout immediately, or have it posted to their own timeline.

The overall design of the site is a black and gold setting with a GUI that follows the one thirds rule. The flow of the site is top to bottom. This is done to be more accessible to mobile devices. Users who are not comfortable with the top to bottom look can change it to left to right in their account preferences. The design of the feedback section is intended to be a direct reflection of the users’ feelings towards a product. Other than feedback being located directly under each product. Users can go to a feedback page where they can see feedback given for each product. It can be sorted either by product, or by the user who gave the review. This page will allow users to reply to other users’ feedback (the replies will not be added to the feedback that is displayed while a user is shopping). This will enable satirical feedback from users who wish to jokingly comment about items available in our shop. While enabling other users who wish to give more serious feedback about purchases a chance to express themselves. This combination of online shopping, and personalized user accounts will hopeful integrate well with todays online community.

The support section will be included because every site needs a support feature. I realize for this class our team will not be able to have a 24/7 support for users. However, the feature will still be implemented as it will give our team experience in creating a more professional website that meets todays standards. Once again while the feature will be there, the actual use of the feature is intended to be minimal and only done to give us experience in todays necessities of online communication.

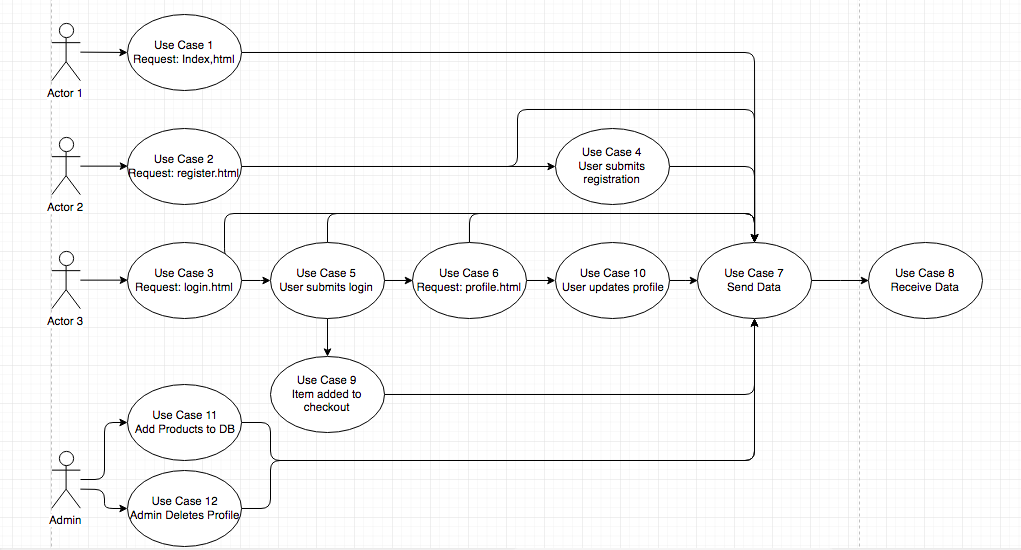
Covering the management side of the project development, we will be using the incremental development model. This is done because no one on our team has experience in constructing a professional style website with features users expect to have. We will be focusing on a single task such as creating a webpage that allows users to register and store information, and getting that part to work before moving on to the next part of the project. This will enable us to find what we are capable of doing, and learn things as we go along. The downside to this will be as we add things change happens and dealing with that change will be difficult. Getting everything to work together will be the main challenge in using the incremental development model.

On a more technical note our team has decided to go with using node js, JavaScript, html, and MongoDB to handle our server/website. Node JS will be used to handle everything server side, and MongoDB will be used to store everything from the products, customers accounts, trending items, and the feedback given. The website will be hosted using Amazon AWS, on an ubuntu server very similar to the one for the tech exercise gone over in class. The reason for this change is the experience the team has in communicating from client to server in JavaScript. Right now we do not plan on using Apache to handle our website or its files and deployment. For everything we have, it is simply not necessary as it can be done with node, JavaScript, and express. It will be easier for us and allow us to spend more time on the design and features of the website, which we are learning is actual quite the colossal task for this semester. We hope to implement everything we have talked about. Right now, we are kind of shooting for the moon, and are developing one thing at a time, and seeing how much we will be able to complete before the semester is over.

1. **Use Cases and Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **Use Case Number** | **Use Case Name** | **Number of Functional Requirements** |
| 1 | Request: Index.html | 5 |
| 2 | Request: register.html | 5 |
| 3 | Request: login.html | 5 |
| 4 | User submits registration | 6 |
| 5 | User submits login | 6 |
| 6 | Request: profile.html | 9 |
| 7 | Send Data | 3 |
| 8 | Receive Data | 3 |
| 9 | Item added to checkout | 6 |
| 10 | User updates profile | 6 |
| 11 | Add product to Db | 3 |
| 12 | Admin Deletes Profile | 4 |

Table 1 - Twelve use cases and their number of functional requirements for OnShop website.

Figure 1. The use case dependency diagram of OnShop shows the use cases for the Client-Server system, the related actors/admins, and the dependency relationships among use cases.

**2.1 Request: index.html**

**2.1.1 Request: index.html documentation**

Use Case Name:

Actors:

Precondition:

Primary Scenario:

* The use case starts when a user goes to the website 18.220.253.158

Secondary Scenarios:

* The use case is also activated when a user goes to 18.220.253.158/location not at our site

Alternate Scenarios:

* None

Exception Scenarios: None

Extension Points:

Used use cases:

Post Conditions: User goes to different page on the website

**2.1.2 Request: index.html Functional Requirements**

1. A request is sent to index.js
2. Index.js then sends the index.html page to the user
3. Index.html then requests the bootstrap, css, and app.js pages
4. Index.js receives the requests and processes them
5. The processed requests are then sent back to the user

**2.2 Request: register.html**

**2.2.1 Request: register.html documentation**

Use case name: Request: register.html

Actors:

Precondition: User not logged in or has no account in database

Primary Scenario:

* The use case starts when user goes to 18.220.253.158/register.html

Secondary Scenario:

* The use case starts from link on 18.220.253.158/login.html

Alternate Scenarios:

Exception Scenarios: User does not enter a valid email address

Extension Points:

Used use cases:

Post conditions: User proceeds to use case 4

**2.2.2 Request: register.html Functional Requirements**

1. A request is sent to index.js
2. Index.js then sends the register.html page to the user
3. register.html then requests the bootstrap, css, and app.js pages
4. Index.js receives the requests and processes them
5. The processed requests are then sent back to the user

**2.3 Request: login.html**

**2.3.1 Request: login.html documentation**

Use case name: Request: login.html

Actors: Users

Precondition: User not logged in and has account in database

Primary Scenario:

* The use case starts when user goes to 18.220.253.158/login.html

Secondary Scenario:

* The use case starts from link on 18.220.253.158/index.html

Alternate Scenarios:

Exception Scenarios: User does not enter a valid login

Extension Points:

Used use cases:

Post conditions: User proceeds to use case 5

**2.3.2 Request: login.html functional requirements**

1. A request is sent to index.js
2. Index.js then sends the login.html page to the user
3. login.html then requests the bootstrap, css, and app.js pages
4. Index.js receives the requests and processes them
5. The processed requests are then sent back to the user

**2.4 User submits registration**

**2.4.1 User submits registration documentation**

Use case name: User submits registration

Actors: Users

Precondition: Use case 2

Primary Scenario:

* The use case starts when use case 2 has been processed

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios: User does not enter a valid email

Extension Points:

Used use cases: use case 2

Post conditions: database processes valid registration

**2.4.2 User submits registration functional requirements**

1. Register.html sends users information to app.js
2. App.js sends information to index.js
3. Index.js submits the query to the database
4. Database processes request and sends result back to index.js
5. Index.js sends result back to app.js
6. App.js processes result and sends result back to register.html

**2.5 User submits login**

**2.5.1 User submits login documentation**

Use case name: User submits login

Actors: Users

Precondition: Use case 3

Primary Scenario:

* The use case starts when use case 3 has been completed

Secondary Scenarios:

Alternate Scenarios:

Exception Scenarios: Invalid login attempt

Extension Points:

Used use cases:

Post Conditions: A valid login has been processed

**2.5.2 User submits login functional requirements**

1. login.html sends users information to app.js
2. App.js sends information to index.js
3. Index.js submits the query to the database
4. Database processes request and sends result back to index.js
5. Index.js sends result back to app.js
6. App.js processes result and sends result back to login.html

**2.6 Request: profile.html**

**2.6.1 Request: profile.html documentation**

Use case name: Request: profile.html

Actors: Users

Precondition: User logged in, use case 5

Primary Scenario:

* Use case 5 is complete and user goes to 18.220.253.158/profile.html

Secondary Scenario:

* Use case 5 is complete and user clicks on link from 18.220.253.158/index.html

Alternate Scenarios:

Exception Scenarios: User goes to 18.220.253.158/profile.html without use case 5 being completed

Extension Points:

Used use cases: Use case 5

Post conditions: User proceeds to use case 10, or goes to another page on website

**2.6.2 Request: profile.html functional requirements**

1. A request is send to index.js
2. Index.js processes request and sends profile.html to user
3. Profile.html sends request for bootstrap, css, and app.js pages
4. Index.js processes request and sends the pages back to the user
5. After page loads app.js sends request to index.js for profile information
6. Index.js sends request to database
7. Database sends result to index.js
8. Index.js sends result to app.js
9. App.js sends result to profile.html

**2.7** **Send data**

**2.7.1 Send data documentation**

Use case name: Send data

Actors:

Precondition:

Primary Scenario:

* Data comes from client and needs to be processed by server

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios: failed to load page, or disconnect/no internet access

Extension Points: database

Used use cases: Use cases: 1,2,3,4,5, 6, 9, 10

Post conditions: Use case 8

**2.7.2 Send data functional requirements**

1. App.js gets information from html page
2. App.js sends information to index.js
3. If needed data is sent to database

**2.8** **Receive data**

**2.8.1 Receive data documentation**

Use case name: Receive data

Actors:

Precondition: use case 7

Primary Scenario:

* Data needs to be sent back to user

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios: failed to load page, or disconnect/no internet access

Extension Points: database

Used use cases: Use case 7

Post conditions: data successfully sent to client

**2.8.2 Send data functional requirements**

1. If needed data from database is sent to index.js
2. Index.js sends data to app.js
3. App.js processes information and sends to html page

**2.9 Item added to checkout**

**2.9.1 Item added to checkout documentation**

Use case name: Item added to checkout

Actors: user

Precondition: use case 7

Primary Scenario:

* Data needs to be sent back to user

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios: failed to load page, or disconnect/no internet access

Extension Points: database

Used use cases: Use case 5

Post conditions: Item successfully added

**2.9.2 Item added to checkout functional requirements**

1. User sends request to app.js to process adding item to checkout
2. App.js sends request to index.js
3. Index.js submits query to database
4. Database sends result of query to index.js
5. Index.js sends result to app.js
6. App.js sends result to user

**2.10 User updates profile**

**2.10.1 User updates profile documentation**

Use case name: User updates profile

Actors: user

Precondition: use case 6

Primary Scenario:

* User enters information in update fields and clicks submit

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios:

Extension Points:

Used use cases: Use case 6

Post conditions: New information successfully processed

**2.10.2 User updates profile functional requirements**

1. User clicks submit and information goes to app.js
2. App.js sends information to index.js
3. Index.js submits query to database
4. Database sends result to index.js
5. Index.js sends result to app.js
6. App.js processes result and sends information back to user

**2.11 Add product to Database**

**2.11.1 Add product to Database documentation**

Use case name: Add product to database

Actors: admin

Precondition:

Primary Scenario:

* Admin adds product to database

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios:

Extension Points:

Used use cases: Use case 6

Post conditions: Product successfully added

**2.11.2 Add product functional requirements**

1. Admin completes form to add product to database and app.js submits data to index.js
2. Index.js sends query to database
3. Database returns results of query

**2.12 Admin Deletes Profile**

**2.12.1 Admin Deletes Profile documentation**

Use case name: Admin deletes profile

Actors: admin

Precondition:

Primary Scenario:

* Admin deletes a profile using a form that submits query to database

Secondary Scenario:

Alternate Scenarios:

Exception Scenarios:

Extension Points:

Used use cases:

Post conditions: Profile successfully deleted

**2.12.2 Admin Deletes Profile Functional Requirements**

1. Admin completes form to delete a specific profile app.js sends information to index.js
2. Index.js submits query to database
3. Database sends result to index.js
   1. Note the result is the entire users profile, this is done to re-add the profile in case of accidental deletion
4. **Non-Functional Requirements**
   1. Must be compatible with Google Chrome.
   2. Must be compatible with Windows 7, Windows 10, Mac OS X, Linux
   3. Javascript must be enabled
   4. Cookies must be enabled
   5. Internet Access is required
   6. Text interfaces are required
   7. The software is closed source to protect the security of the users.
   8. Registration is required to use all features of the website.
   9. Email is required for registration.
   10. The OnShop store will allow users to comment on products.
   11. Passwords shall never be viewable at the registration/login page.
   12. Username and passwords cannot be duplicates
   13. Passwords must be at a minimum 6 characters, and include a number.
   14. The OnShop store does not guarantee that the Help Center answers will solve all issues users encounter.
   15. Must agree to terms and conditions before registration and purchase.

**References**

Style and front end of website: <https://github.com/twbs/bootstrap>

Server side: <https://github.com/nodejs/node>

Client-Server communication: <https://github.com/expressjs/express>

Database: <https://github.com/mongodb/mongo>

**Appendix**

Andrew Vaughn : Table of Contents, Functional/Non-Functional Requirements

Stephen Knofczynski : Case Dependency Diagram, Functional/Non-Functional Requirements

Brian Hawkins : Layout/Design, Functional/Non-Functional Requirements