Software Engineering- J-Component

Final Report

UrDoorBoT

By
ATLURI BHUMIKA (19BDS0109)
NIKHITHA PERAPOLA (19BDS0125)
SHAIL PATEL TEJAS (19BCE2698)

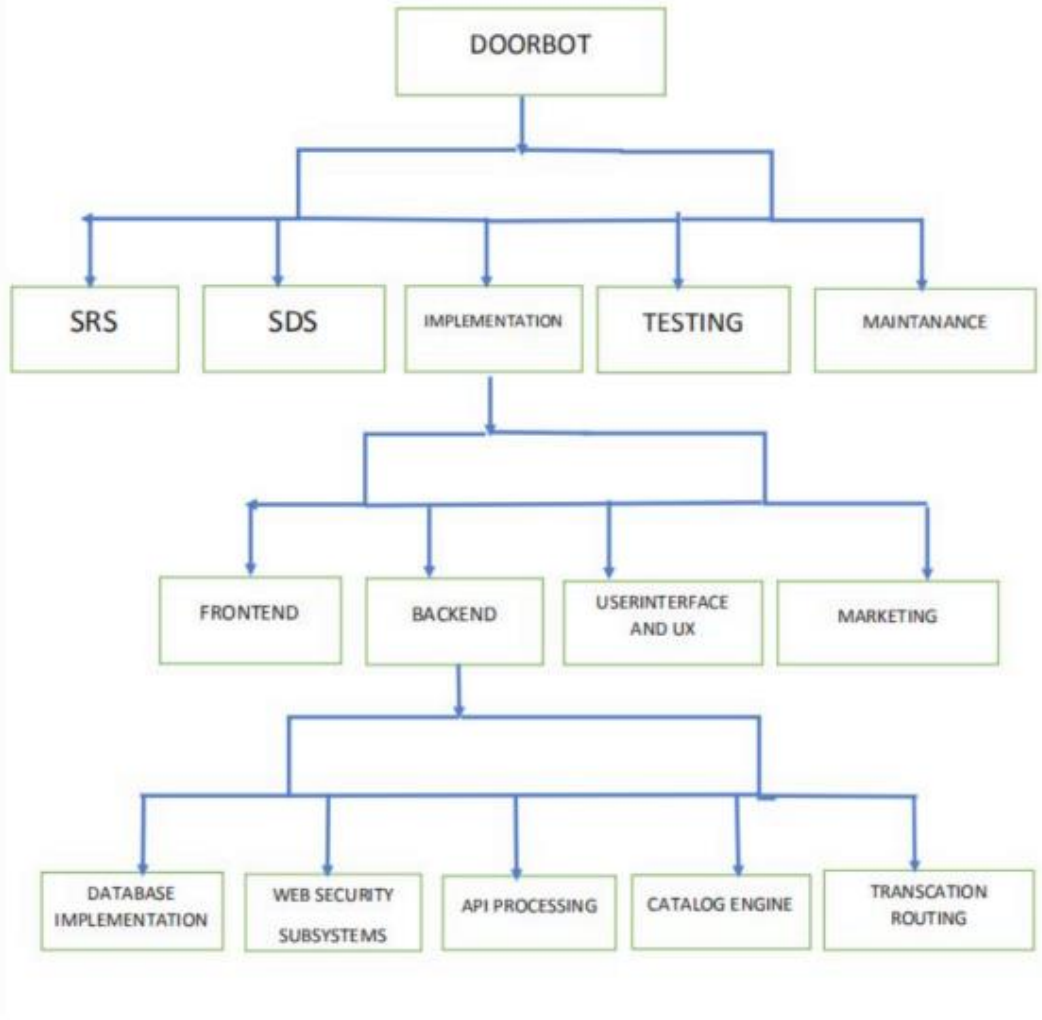
Project Description

Our Project UrDoorBot will be a service-based platform for all small to large scale retailers and shops based on local delivery system for purchases of days to day requirements like groceries. This can be extended to all other exchange services. The home stuck customers can use our website to access all the stores and contact them enabling a specific place shopping system rather than product-based shopping.

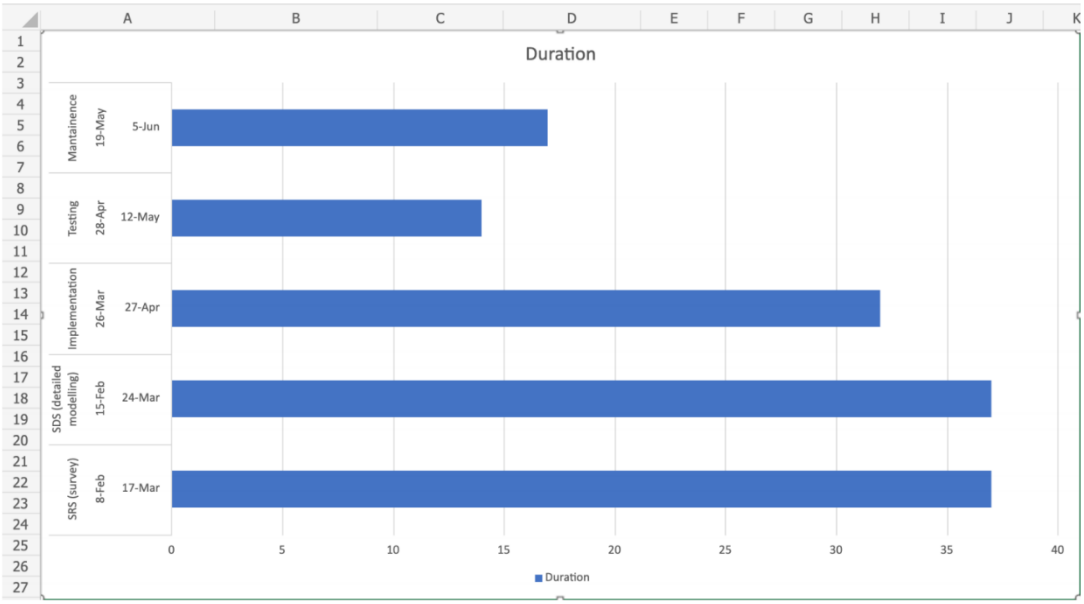
Process Model

We will be using the Spiral Model as a Process Model. We have used the spiral model because the spiral model allows us to handle the risks and allows us to be flexible with our work while depending on the customer satisfaction. As our Software is fully consumer and customer based, this model helps here considering the other models. Other major models like Waterfall Model, RAD model and prototyping model do not allow the same amount of flexibility, and the feedback loop (iterative phases) which makes the other models incompatible with our project.

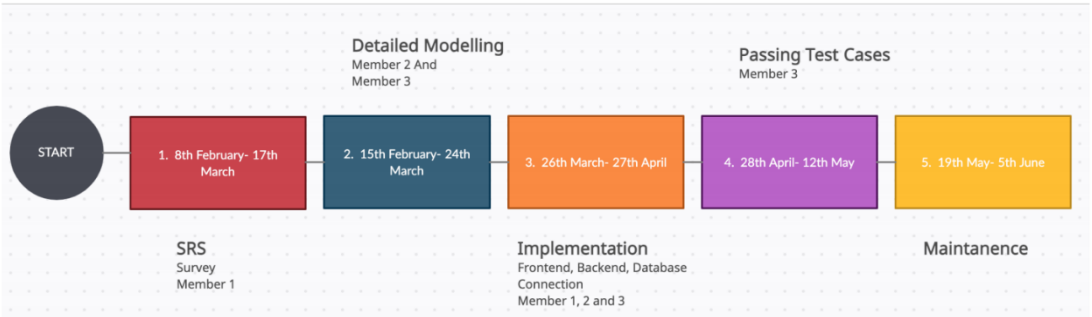
WBS


DOORBOT
SRS
SDS
IMPLEMENTATION
TESTING
MAINTANANCE
FRONTEND
BACKEND
USERINTERFACE AND UX
MARKETING
DATABASE IMPLEMENTATION
WEB SECURITY SUBSYSTEMS
API PROCESSING
CATALOG ENGINE
TRANSCATION ROUTING

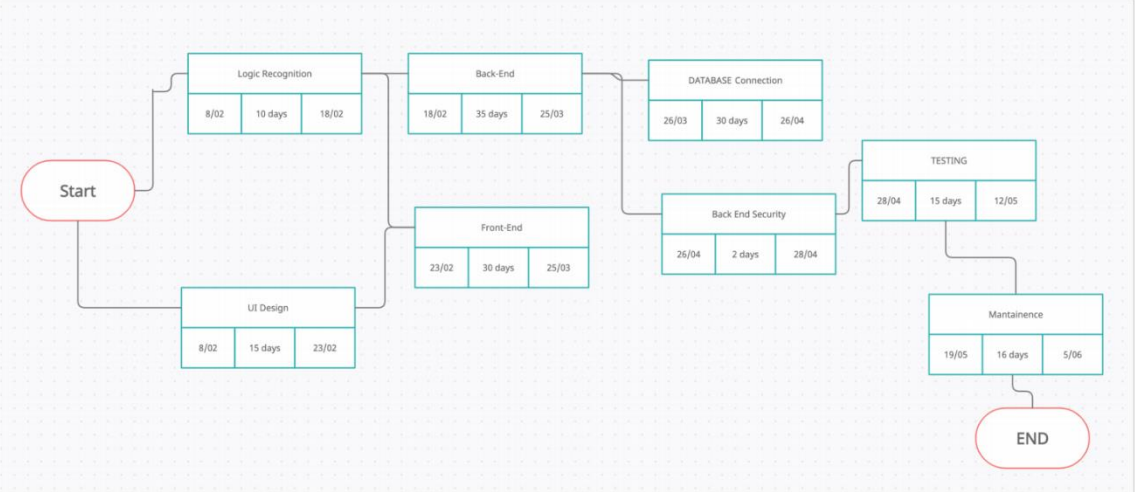
Gantt chart



Timeline chart



PERT


Start
Logic Recognition
8/02
10 days
18/02
Back-End
18/02
35 days
25/03
DATABASE Connection
26/03
30 days
26/04
TESTING
28/04
15 days
12/05
Front-End
23/02
30 days
25/03
Back End Security
26/04
2 days
28/04
UI Design
8/02
15 days
23/02
Mantainence
19/05
16 days
5/06
END

Software Requirements Specification

UrDoorBoT

Table of Contents

Revision History

NAME	DATE	Reason for Changes	Version
SRS 1.0	15th April 2020		1

1.INTRODUCTION:

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references and overview of the SRS. The aim of this document is to gather and analyze and give an in-depth insight of the complete DOOR BOT software system by defining the problem statement in detail.

1.1 PURPOSE

Door bot is an E-commerce website that allows small shops and local retailers to supply goods locally with no barriers of time or distance. This website allows vendors as well as customers to exchange goods through a single get away using the internet. This website boosts up local suppliers and shops income in their area and increases the efficiency of supplying goods within a very less time. The Database administrator will approve and reject requests based on the algorithms designed and consensus designed by the developers to provide guarantee of their supplies and trust of the supplier.

1.2 Document conventions:

SRS-Software requirement specification GUI-
graphical user interface
CRM-customer relationship management
FAQ-Frequently asked questions
STAKEHOLDERS-the persons who participate in the system.

1.3 Scope:

The scope of the project is to provide a commercial transaction between the customer and the supplier .As it is a decentralized system the administrator and the maintenance team provides the centralized functionality .This website allows customers to depend completely on the local facilities (shops) which boosts the GDP of the country and increases customers profit when compared to other markets outside.

1.4: References

Reading about E-commerce website https://en.wikipedia.org/wiki/E- commerce
White paper of www.Amazon web services.
Functional diagrams on https://creately.com/diagram/example/hdkb50lr/Flipkart+DFD

User interface and prototype by http://www.uianduxdesign.com/

1.5: Overview:-

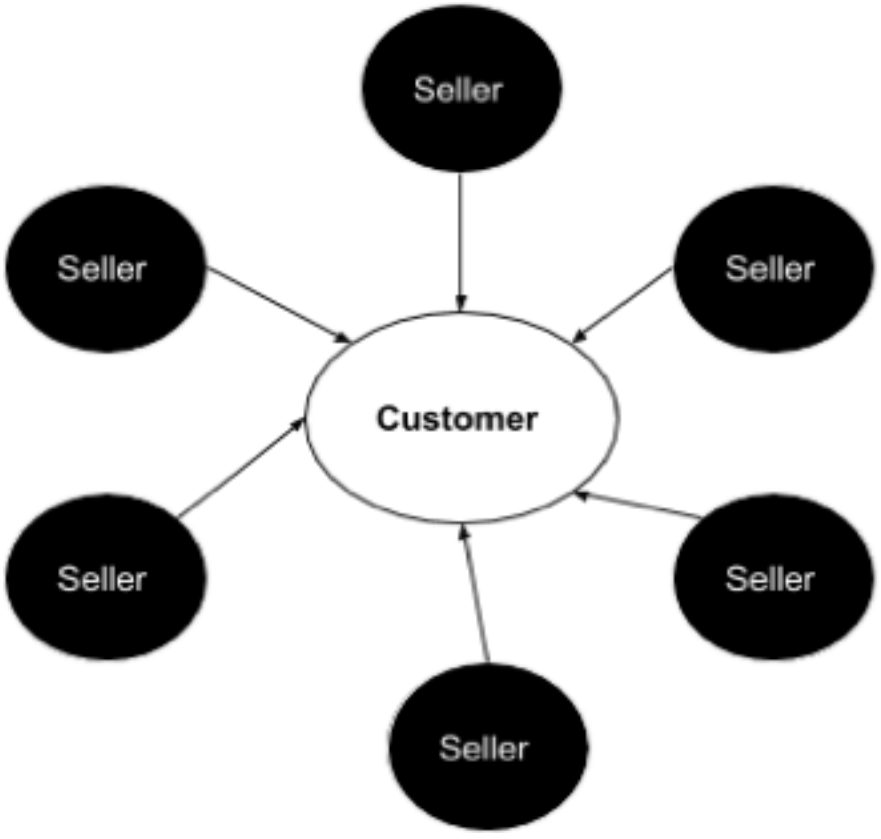
This system provides a portable solution to all those affected by large commercial market. The remaining sections of this document provide a general description, including characteristics of the users of this project, the product's hardware, and the functional and data requirements of the product. . General description and understanding of the project is discussed in section 2 of this document. Section 3 gives the functional requirements, data requirements and assumptions made while designing the website.

2. Overall Description

2.1 Project Perspective

We will be using the Spiral Model as a Process Model. We have used the spiral model because the spiral model allows us to handle the risks and allows us to be flexible with our work while depending on the customer satisfaction. As our Software is fully consumer and customer based, this model helps here considering the other models. Other major models like Waterfall Model, RAD model and prototyping model do not allow the same amount of flexibility, and the feedback loop(iterative phases) which makes the other models incompatible with our project.

2.2 Project Functions



The project functions by allowing the customer(shopper) to access and contact the various shopkeepers(sellers) directly around his place through a huge database collection, containing the details of all the various customers.

3. External Interface Requirements

3.1 User Interfaces

The home screen will show the list of all nearby shops within the given range. The users can click on the shop to see the list of items present in that particular shop. The users will also be given an option to contact the shop owner and tell them the required items they want .
· FRONT-END TECHNOLOGY – HTML, CSS, JAVASCRIPT

· BACK-END TECHNOLOGY – nodeJS · DATABASE TECHNOLOGY - mySQL

3.2 Hardware Interfaces

Since the web portal does not have any designated hardware, it does not have any direct hardware interfaces. The GPS required for getting the user's location is managed by the GPS software in the user's system , the user will be asked to give permission for accessing the GPS software in their device after which the shops based on their locality will be shown.
The hardware connection to the database server is managed by the underlying operating system on the mobile phone and the web server.

· PROCESSOR – DUAL CORE
· RAM – MINIMUM 2GB

Web Server Deployment and Technologies	
Nginx	**Nginx** (pronounced as "engine X") is a lightweight open source web server developed by Igor Sysoev.
MySQL	**MySQL** database for storage of Data and user as well as seller information
RESTful API	A **RESTful API** is an application program interface (**API**) that uses HTTP requests to GET, PUT, POST and DELETE data.

3.3 Software Interfaces

The users have to allow the access permission for the GPS software in their device so that they can get the results according to their locality . The software is web-based so a compatible browser is required with stable internet connection .
· OS – WINDOWS, LINUS , MACOS , ANDROID , IOS

· DATABASE - mySQL

3.4 Communications Interfaces

Users can interact with the stores and the owners using the browser itself after they have successfully logged in.

4.SYSTEM FEATURES: -

4.1 Priority:

Medium priority is concerned when it comes to component ratings such as benefit,cost,risk,penalty .As these properties concern after analysis we will analyse and move to the next step of the project .

4.2 Stakeholders:

When we talk about the requirements and development the following people such as customers,suppliers,administrators come all the way throughout the project and act as a key elements in enchantment of the system .

4.3 Functional requirements:

Various user interfaces are designed to facilitate user to interact:-

- ● Login in or sign up page
- ● Registration details
- ● A web page which describes the available goods in the shop in your local area .Option
  will be provided to select location within a certain distance
- ● Shopping cart and transaction page is separately displayed after processing the goods
  to your cart.
- ● After completion of payment the order details will be sent through message to customer
  and the supplier.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

In order to make the user experience good we have very good servers so that if many users at one time send HTTP requests the server won't stop. The users can make their item wish list anytime they want the server will be up 24x7
The main purpose of the normalization is to reduce redundancy which means that data will only be stored once. Repeated storage leads to data depletion and growth over the full size of stored data.

If the database is not done properly it can trigger a diversion. An inconsistency of change occurs when data is added to, modified or removed from the data table. Similarly, in traditional data and incorrectly created data, data duplication data can be problematic. This can be solved by customizing the database.

5.2 Safety Requirements

In the case of any kind of damage in the database like disk failure or system crash a lot of data can be lost so to avoid it we will have a backup storage .

5.3 Security Requirements

Security will be in our top priority , user's login data or any kind of data will be secured very well. The communication between the owner and the user will be end to end encrypted.
Not every shop owner will be allowed to make their account owners with proper documents of the shops will only be allowed.

5.4 Software Quality Attributes

Availability – The portal will be up for 24x7 anytime a user can use the software Maintainability – The software will be maintained time to time Usability – The shop type should satisfy a maximum number of users.

5.5 Business Rules

Not Applicable

Software Design Specification

And

Prototyping with GUI.

UrDoorBoT

Table of Contents

Already Mentioned in SRS document

Revision History

Version	Name	Reason For Changes	Date
1.0	ATLURI BHUMIKA NIKHITHA PERAPOLA SHAIL PATEL TEJAS	*Initial Revision*	*15TH April 2021*

Approved By

Name	Signature	Department	Date

1. Introduction

1.1 Purpose

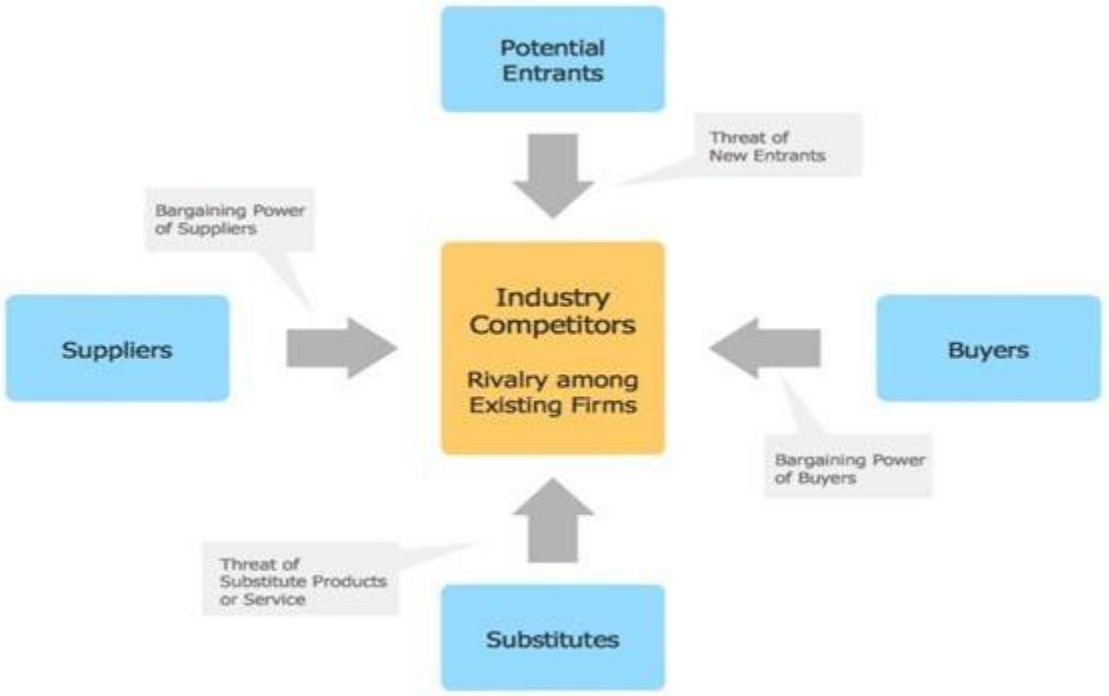
UrDoor bot is an E-commerce website that allows small shops and local retailers to supply goods locally with no barriers of time or distance. This website allows vendors as well as customers to exchange goods through a single get away using the internet. This
website boosts up local suppliers and shops income in their area and increases the efficiency of supplying goods within a very less time. The Database administrator will approve and reject requests based on the algorithms designed and consensus designed by the developers to provide guarantee of their supplies and trust of the supplier.

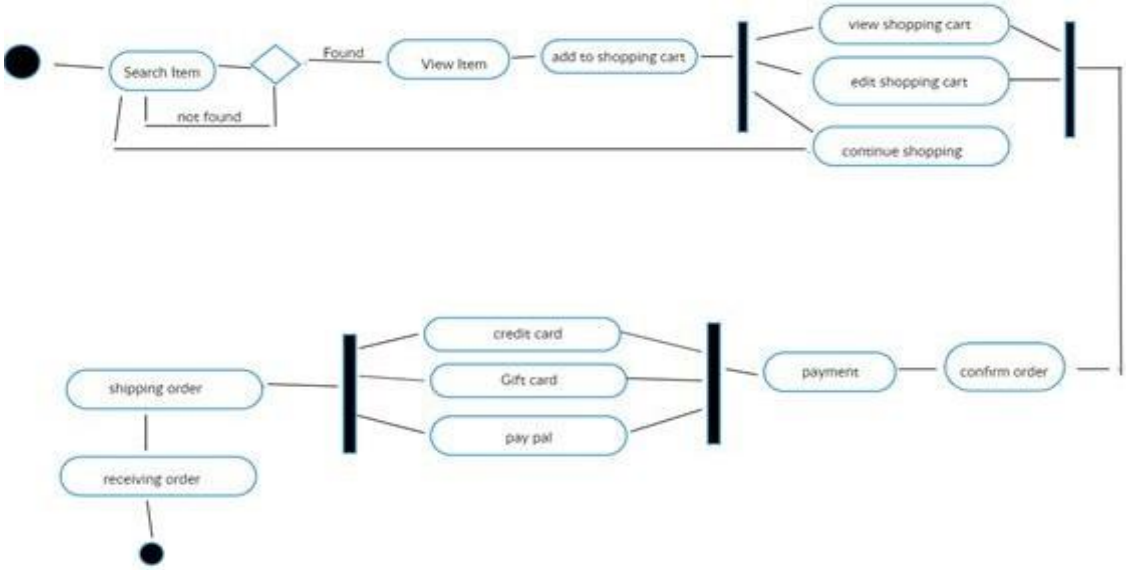
1.2 System Overview and Scope

The scope of the project is to provide a commercial transaction between the customer and the supplier. As it is a decentralized system the administrator and the maintenance team provide the centralized functionality. This website allows customers to depend completely on the local facilities (shops) which boosts the GDP of the country and
increases customers profit when compared to other markets outside.

1.3 Design Map

Block Diagram


Potential Entrants
Threat of New Entrants
Bargaining Power of Suppliers
Suppliers
Industry Competitors
Rivalry among Existing Firms
Buyers
Bargaining Power of Buyers
Threat of Substitute Products or Service
Substitutes


Search Item
Found
not found
View Item
add to shopping cart
view shopping cart
edit shopping cart
continue shopping
credit card
Gift card
pay pal
payment
confirm order
shipping order
receiving order

2. Design Considerations

2.1 Assumptions And Dependencies

• The design of the web application involves the design of the forms for listing the products, search for products, display the complete specification for the product, and design a shopping cart that is easy to use.

• We assume and Design of an interactive application that enables the user to filter the products based on different parameters.

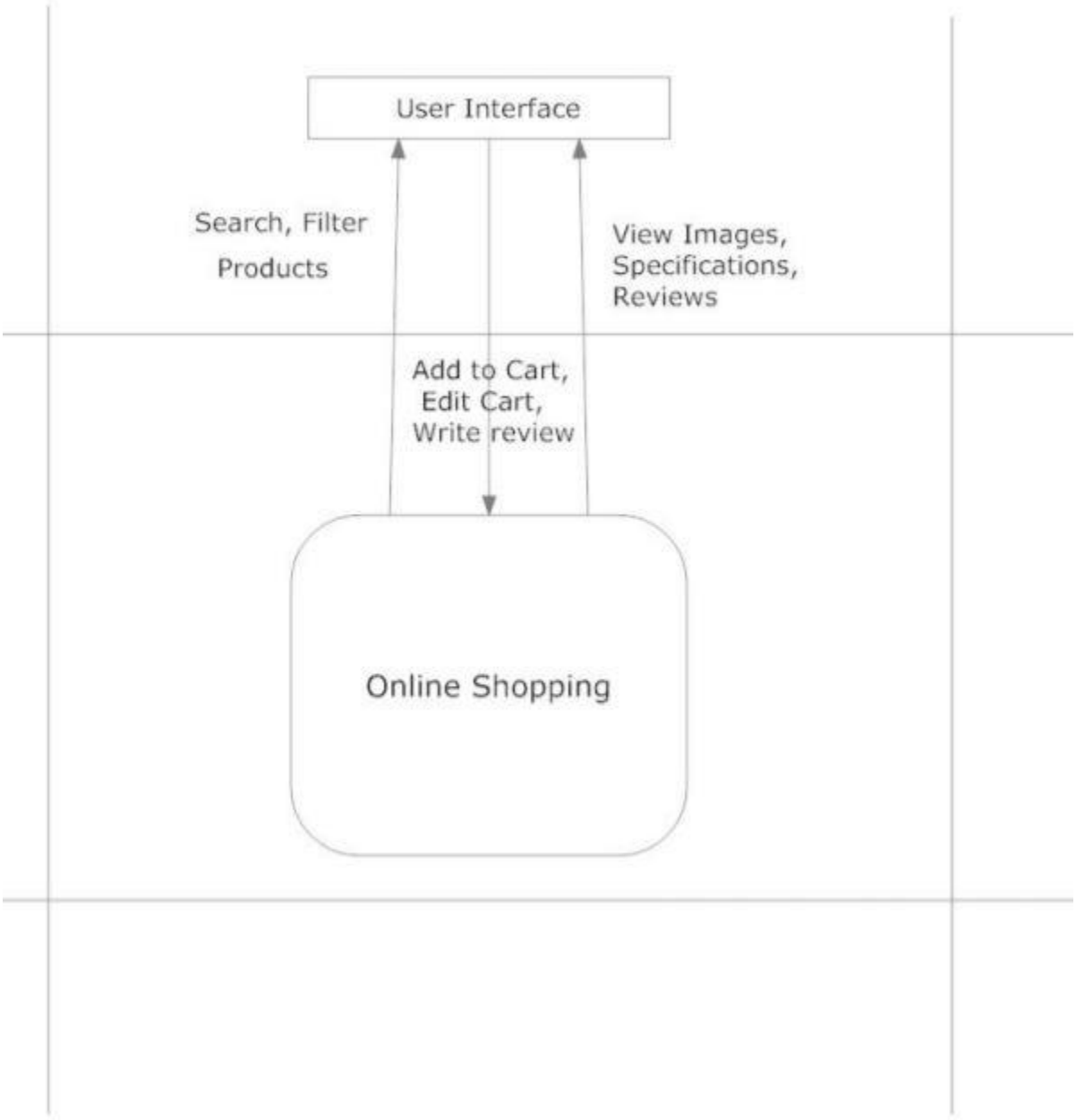
• We consider Design of an application that has features like drag and drop etc.

Design of application that decreases data transfers between the client and the server.

2.2 Constraints

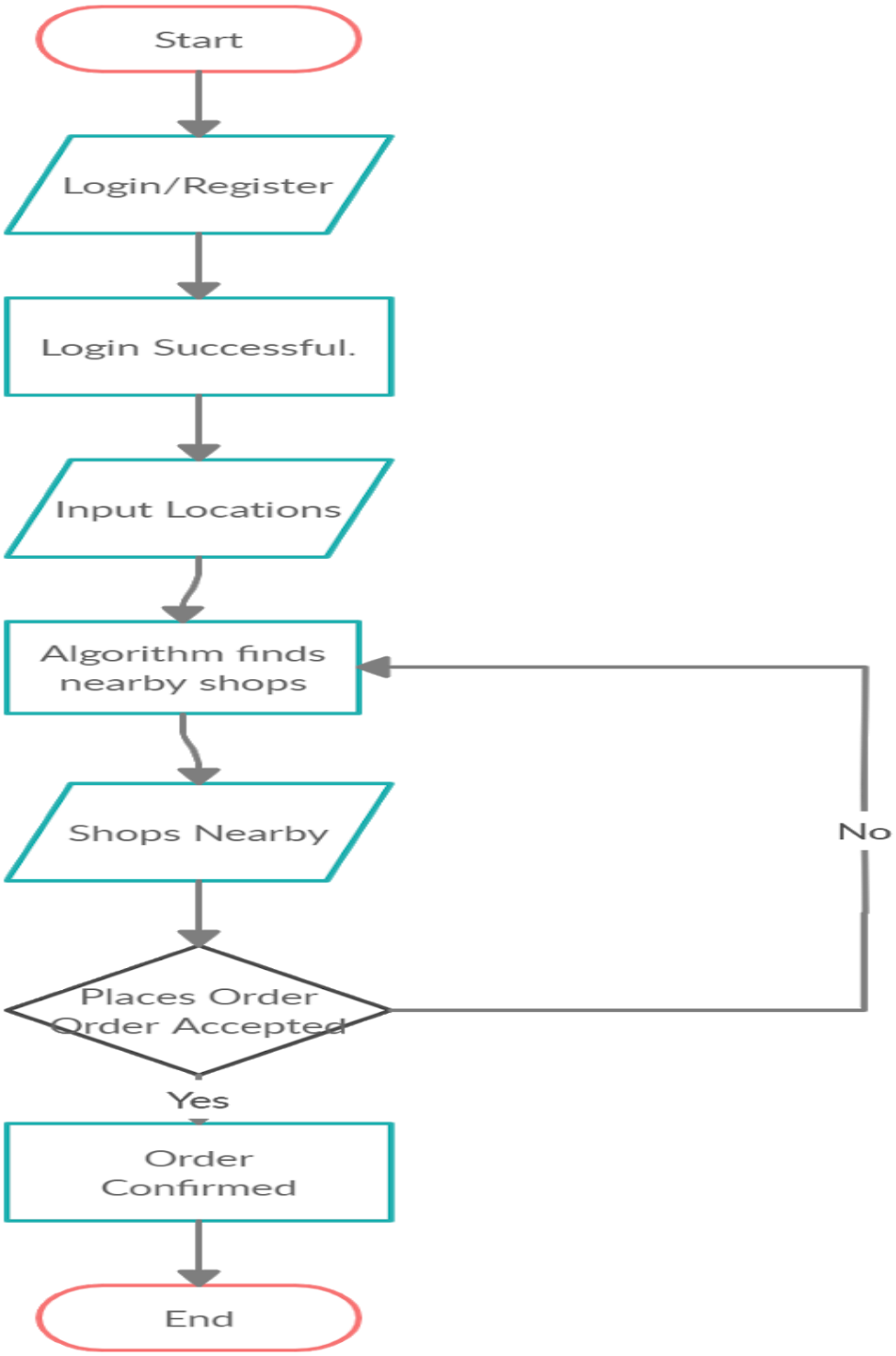
In order to create good software, design constraints and other contingencies need to be addressed. In this section we will enumerate any and all considerations that must be
made when creating this project. It Must be coded efficiently enough to run well on provided server hardware Client-side code and/or web pages must be able to run efficiently on low end hardware. The database will be created and maintained in a way that makes it of reasonable and manageable size.

2.3 System environment


User Interface
Search, Filter
Products
View Images,
Specifications,
Reviews
Add to Cart,
Edit Cart,
Write review
Online Shopping

2.4 Design Methodology

Algorithm:



3. Architecture

3.1 Overview

Not Available

3.2 Subsystem, Component, or Module 1 …N

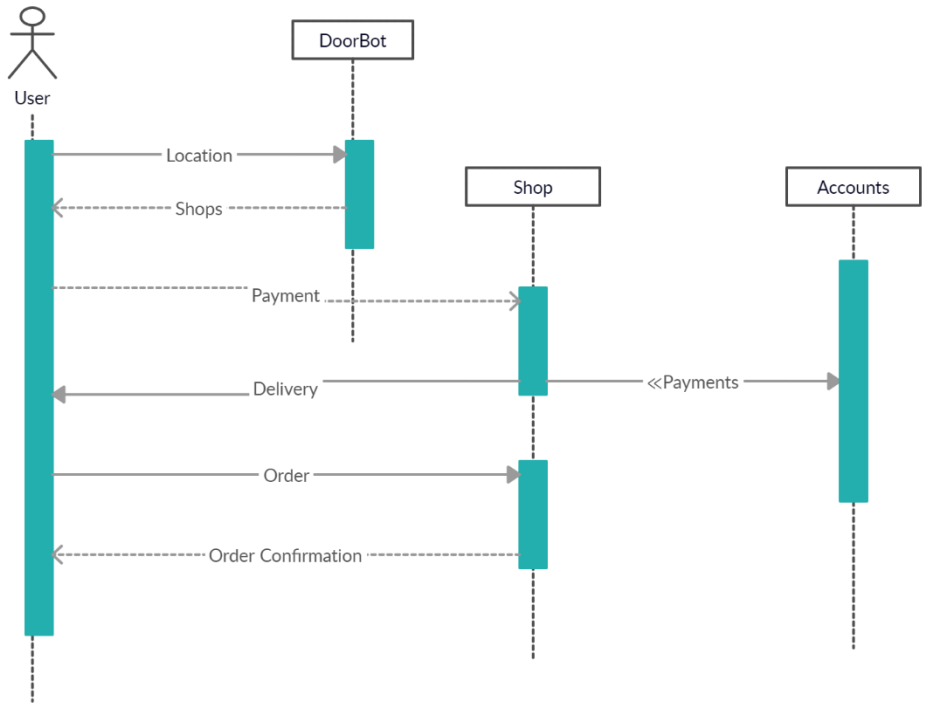
Not Available

3.3 Strategy 1…N

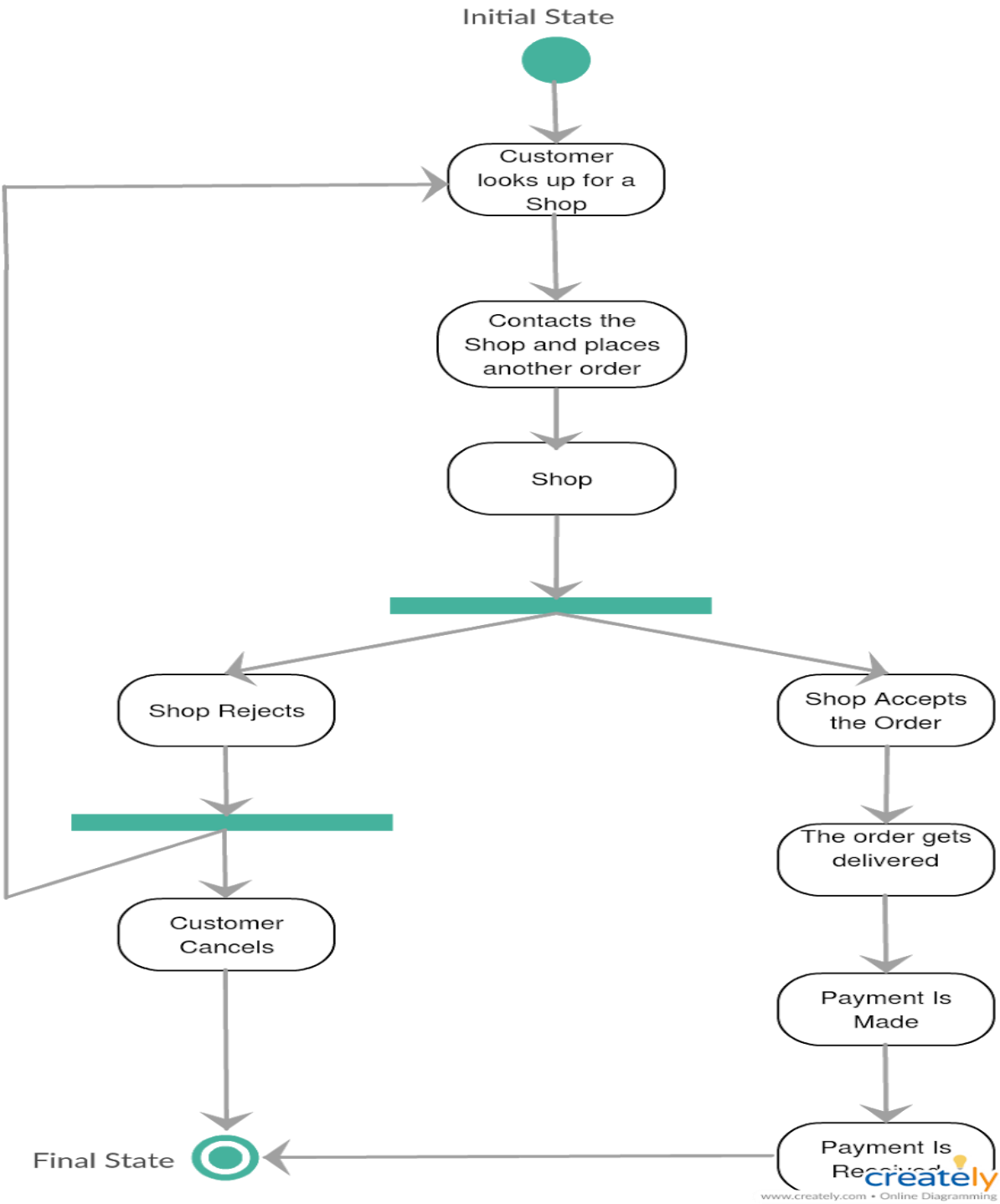
Not Available

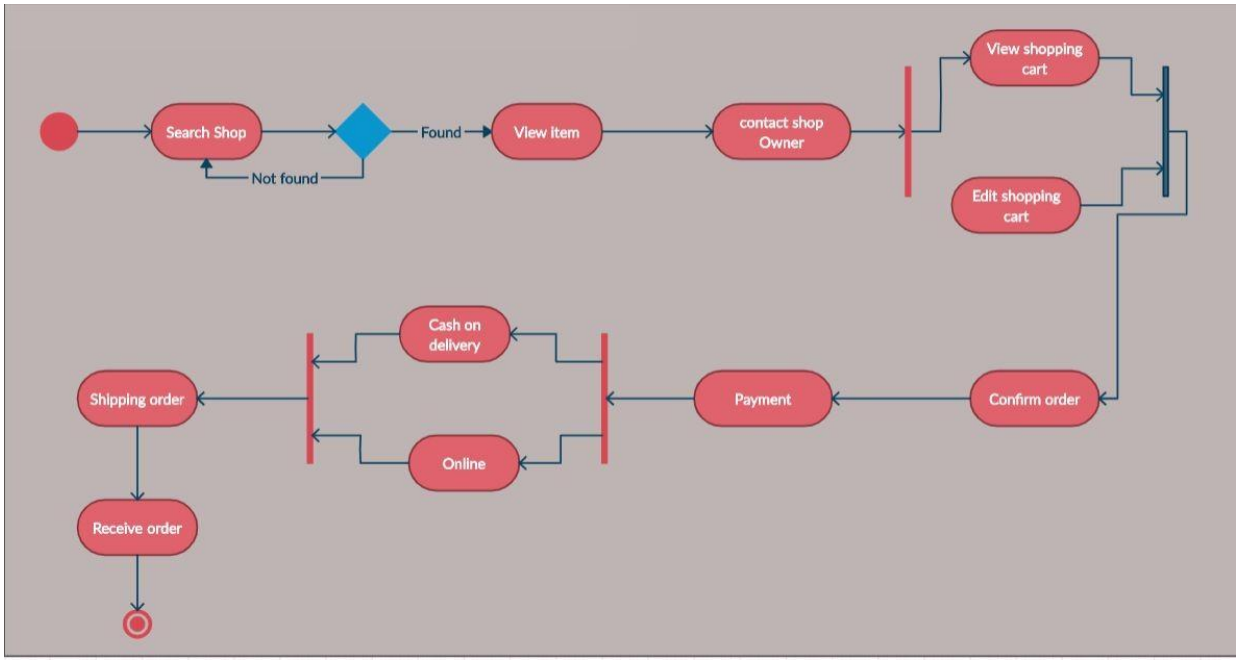
3.4 UML Models

Sequence Diagram

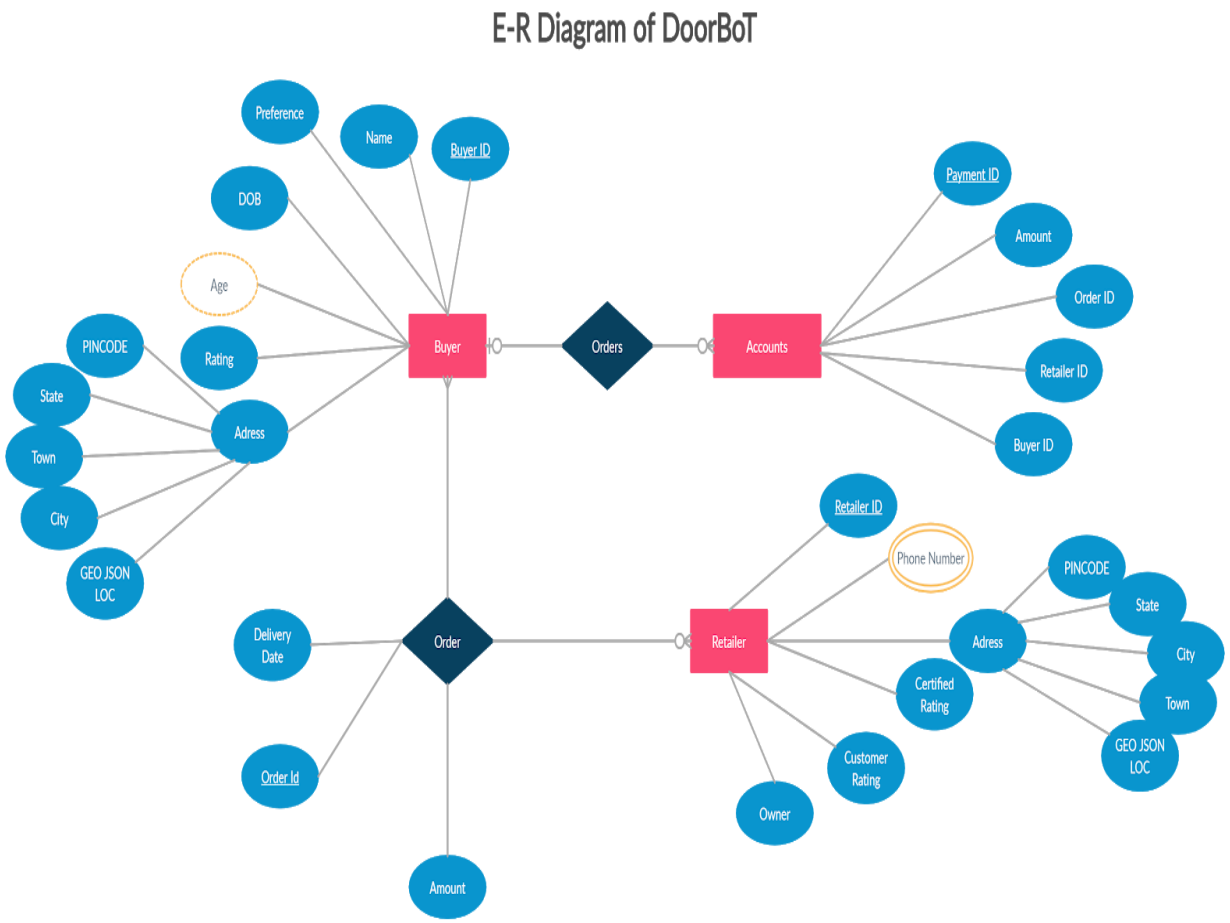


State Chart


Initial State
Customer looks up for a Shop
Contacts the Shop and places another order
Shop
Shop Rejects
Shop Accepts the Order
The order gets delivered
Customer Cancels
Payment Is Made
Final State
Payment Is
creately
www.creately.com • Online Diagramming


Search Shop
Found
Not found
View item
contact shop Owner
View shopping cart
Edit shopping cart
Confirm order
Payment
Cash on delivery
Online
Shipping order
Receive order

4. Database Schema



4.1 Tables, Fields and Relationships

Not Available

4.2 Data Migration

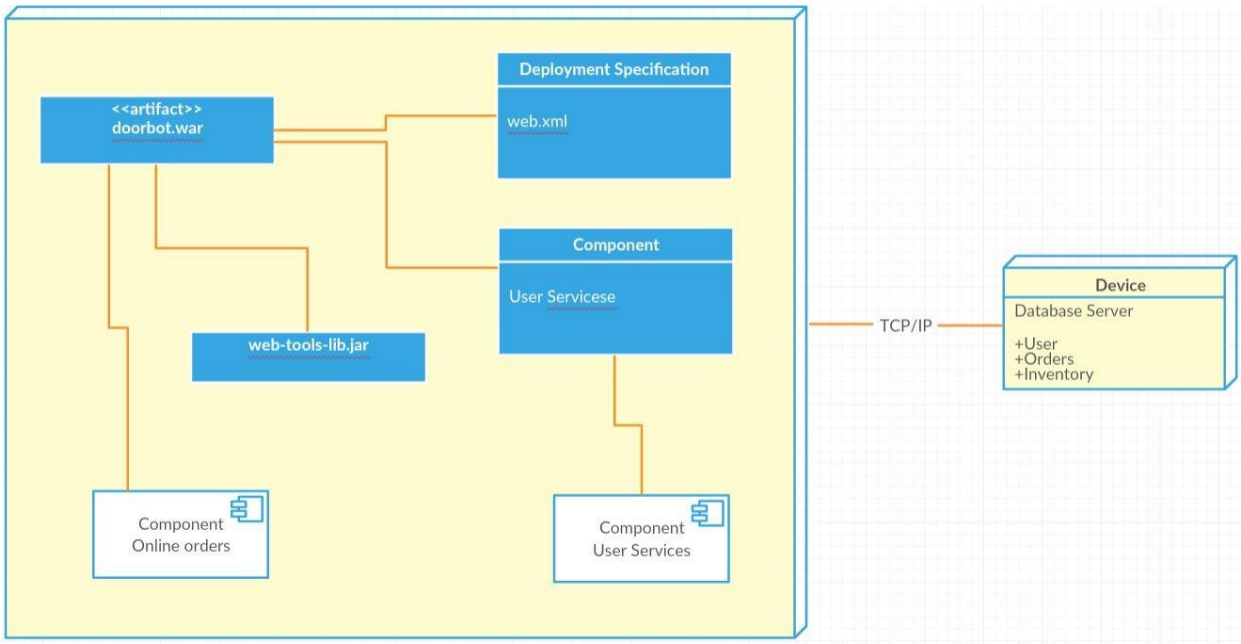
Not Available

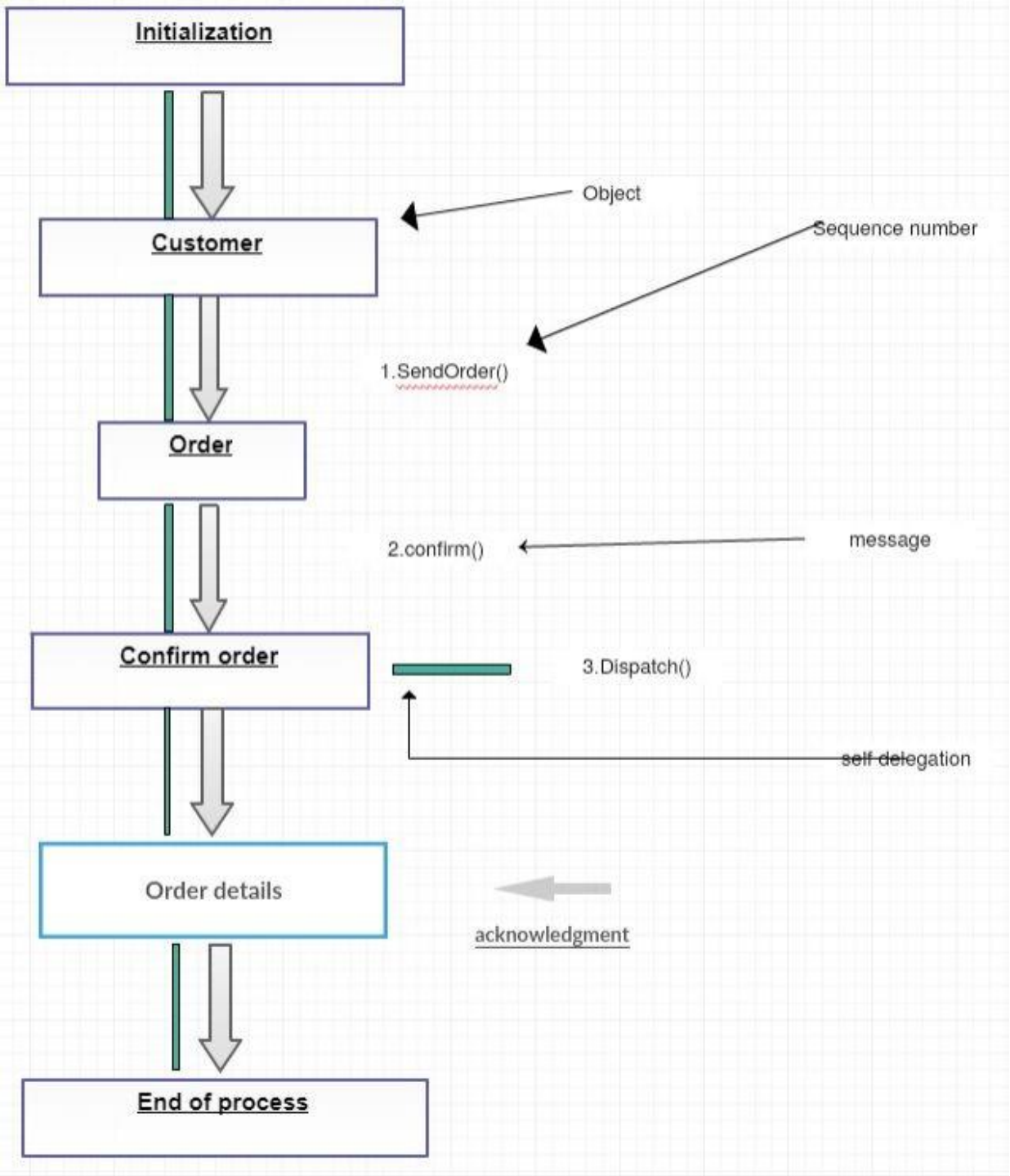
5 . High Level Design

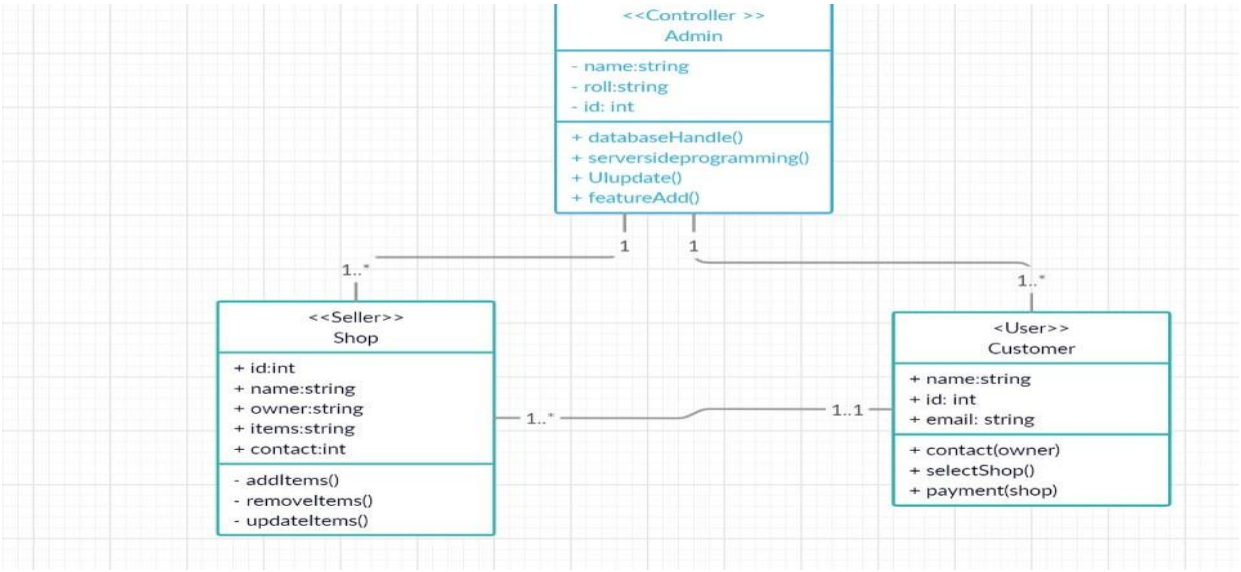
Not Available

6 .Low Level Design

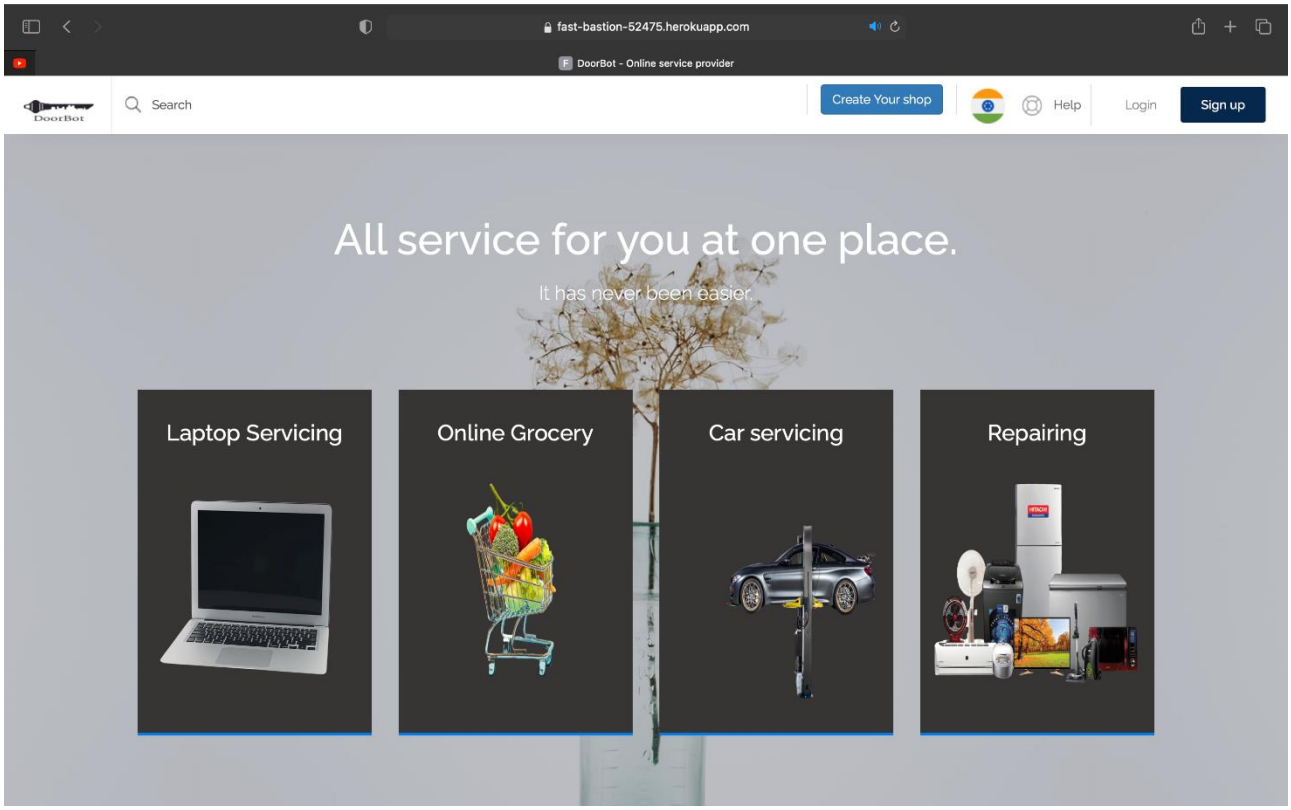
Not Available

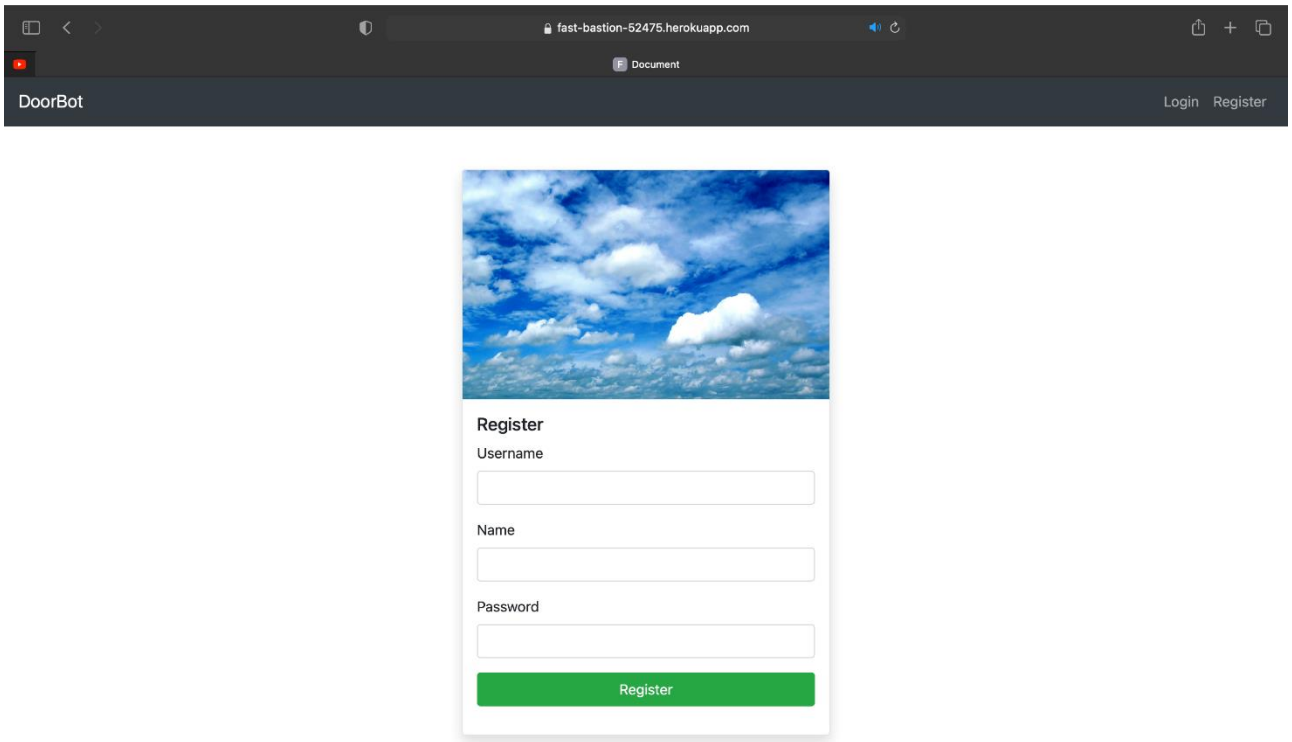


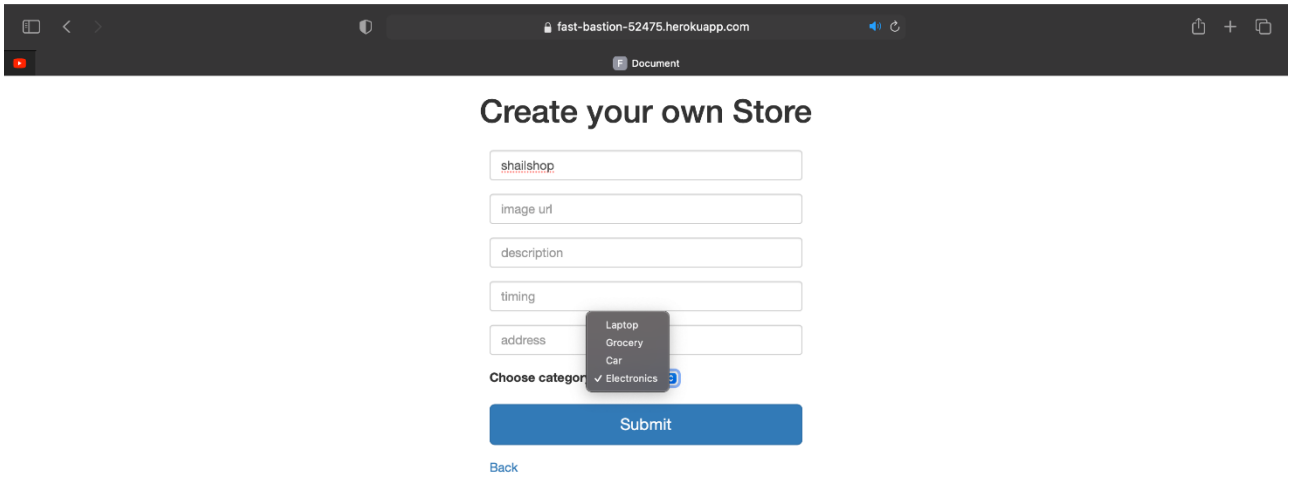

Initialization
Customer
Object
Sequence number
1.SendOrder()
Order
2.confirm()
message
Confirm order
3.Dispatch()
self delegation
Order details
acknowledgment
End of process

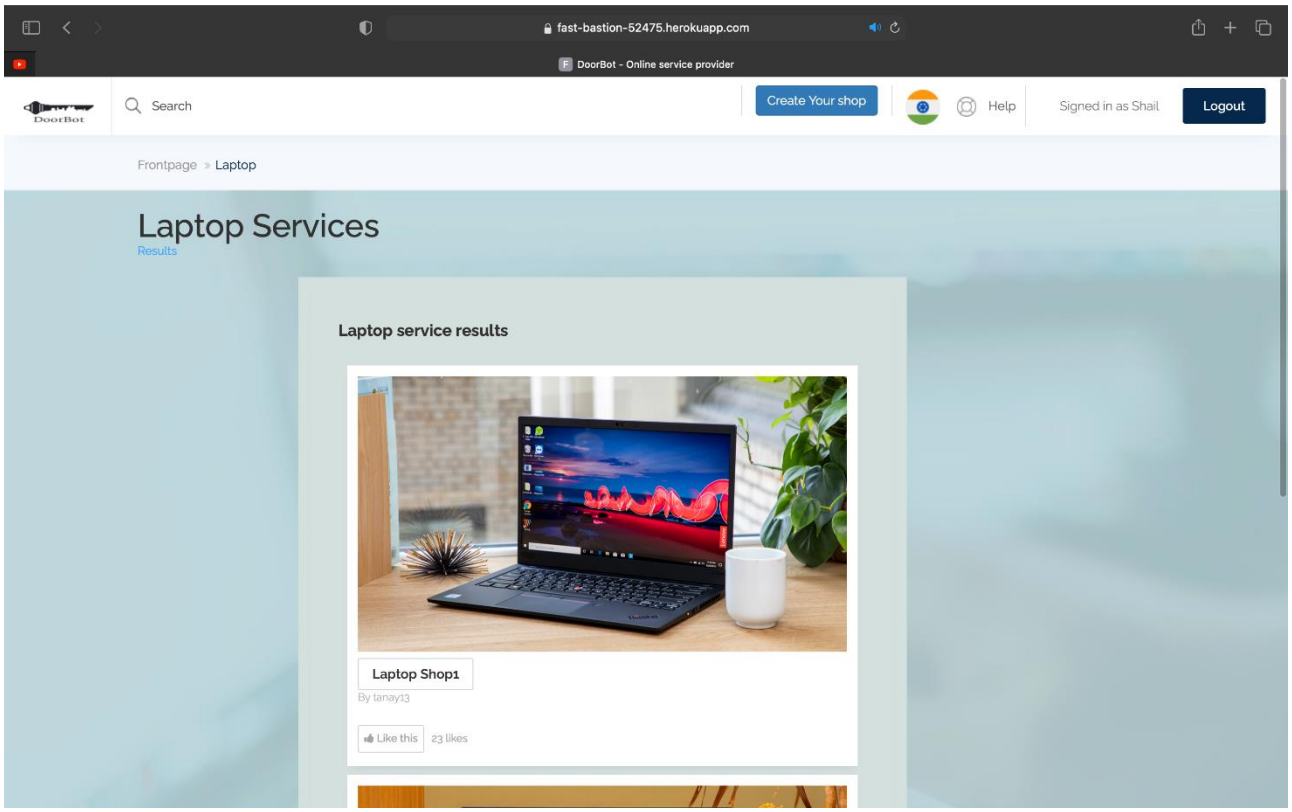

<<Controller >>
Admin
- name:string
- roll:string
- id: int
+ databaseHandle()
+ serversideprogramming()
+ Ulupdate()
+ featureAdd()
1
1
1..*
1..*
<<Seller>>
Shop
+ id:int
+ name:string
+ owner:string
+ items:string
+ contact:int
- addItems()
- removeItems()
- updateItems()
1..*
1..1
<User>>
Customer
+ name:string
+ id: int
+ email: string
+ contact(owner)
+ selectShop()
+ payment(shop)

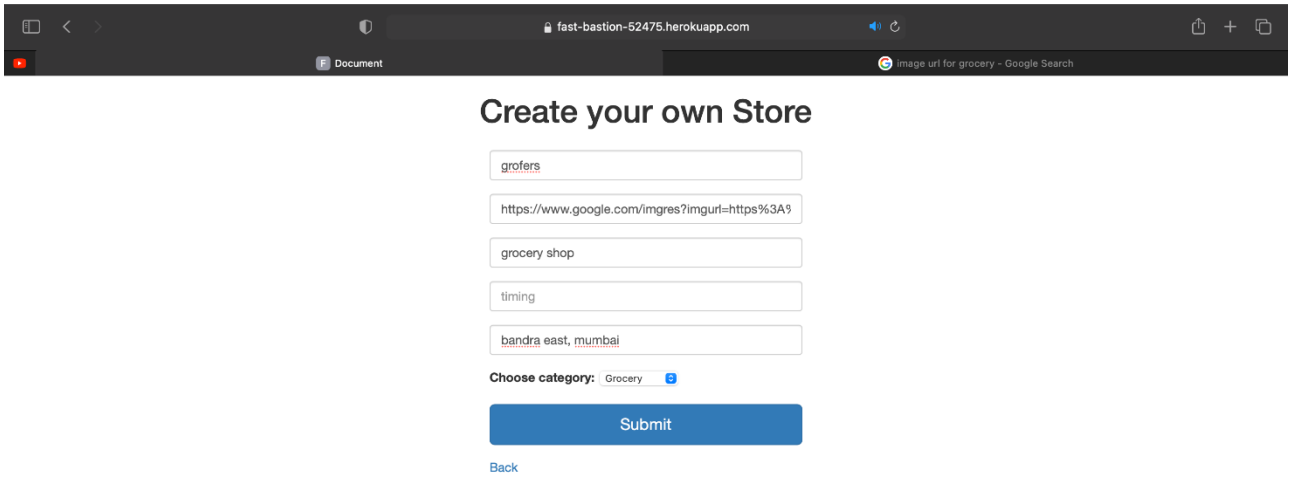
USER INTERFACE DESIGN

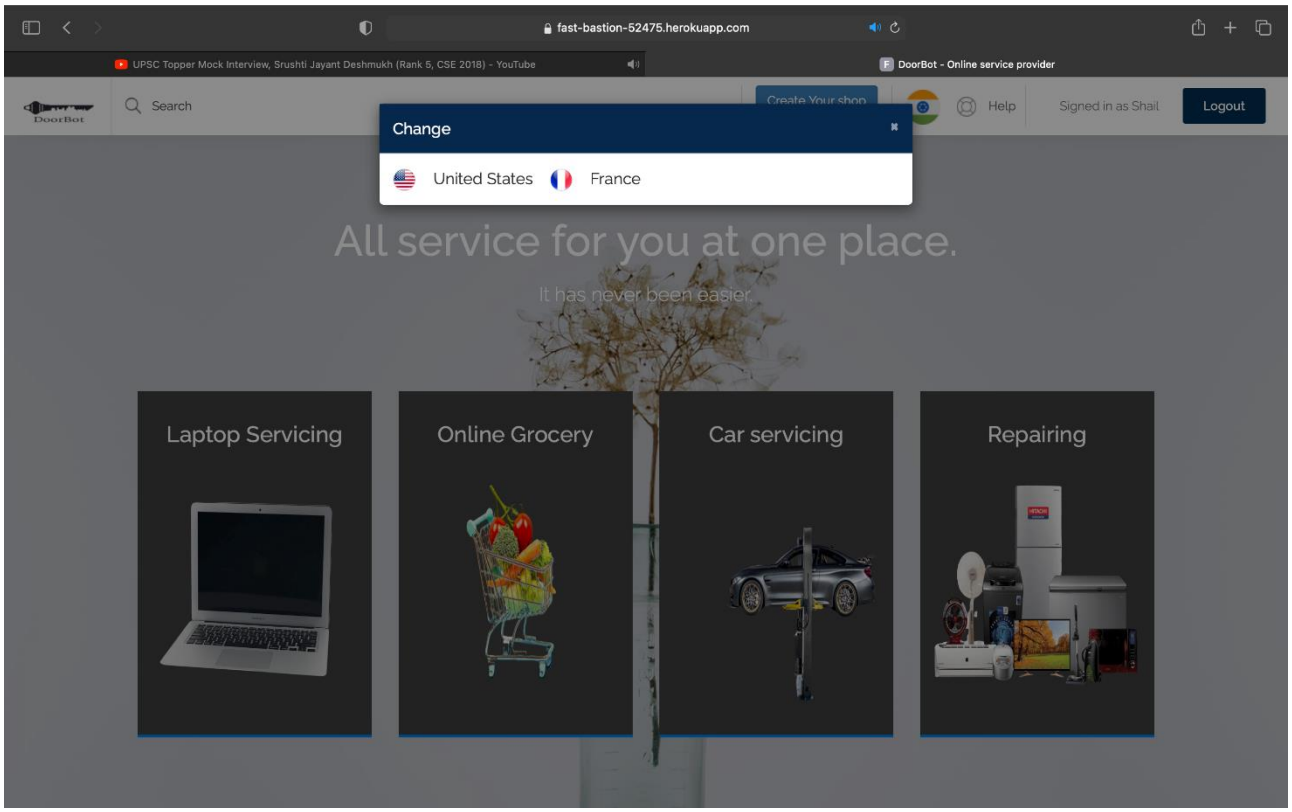



fast-bastion-52475.herokuapp.com
Document
DoorBot
Login
Register
Register
Username
Name
Password
Register


fast-bastion-52475.herokuapp.com
Document
Create your own Store
shailshop
image url
description
timing
address
Laptop
Grocery
Car
Electronics
Choose categor
Submit
Back


fast-bastion-52475.herokuapp.com
DoorBot - Online service provider
DoorBot
Search
Create Your shop
Help
Signed in as Shail
Logout
Frontpage » Laptop
Laptop Services
Results
Laptop service results
Laptop Shop1
By tanay13
Like this
23 likes


fast-bastion-52475.herokuapp.com
Document
image url for grocery - Google Search
Create your own Store
grofers
https://www.google.com/imgres?imgurl=https%3A%
grocery shop
timing
bandra east, mumbai
Choose category:
Grocery
Submit
Back



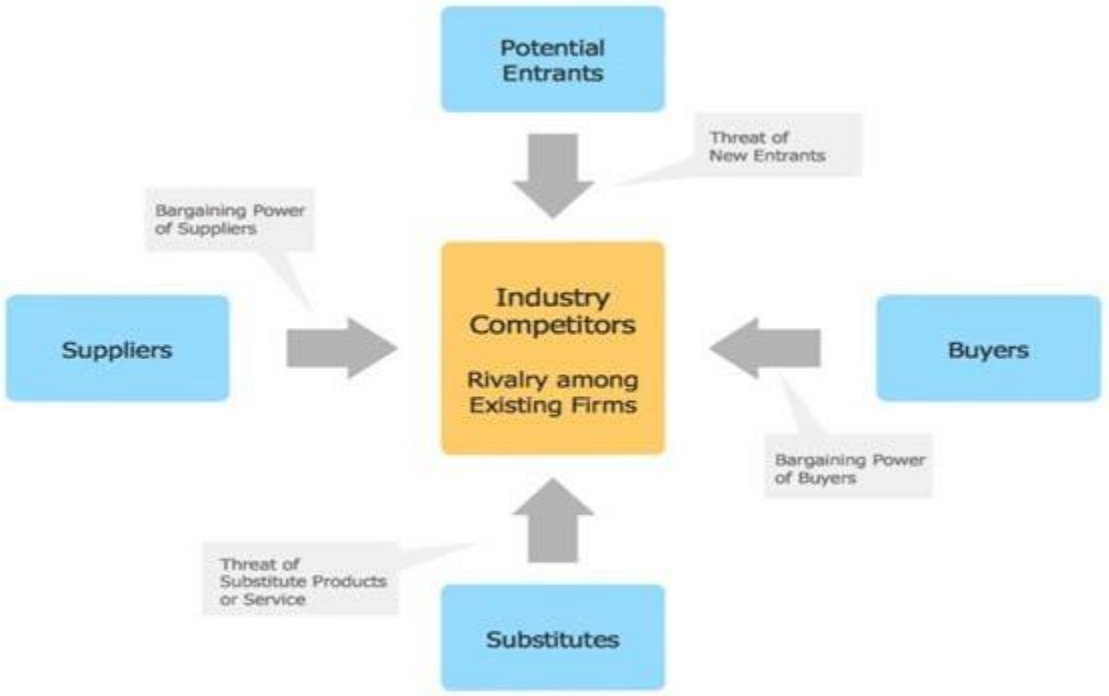
7 Appendix A: Project Timeline

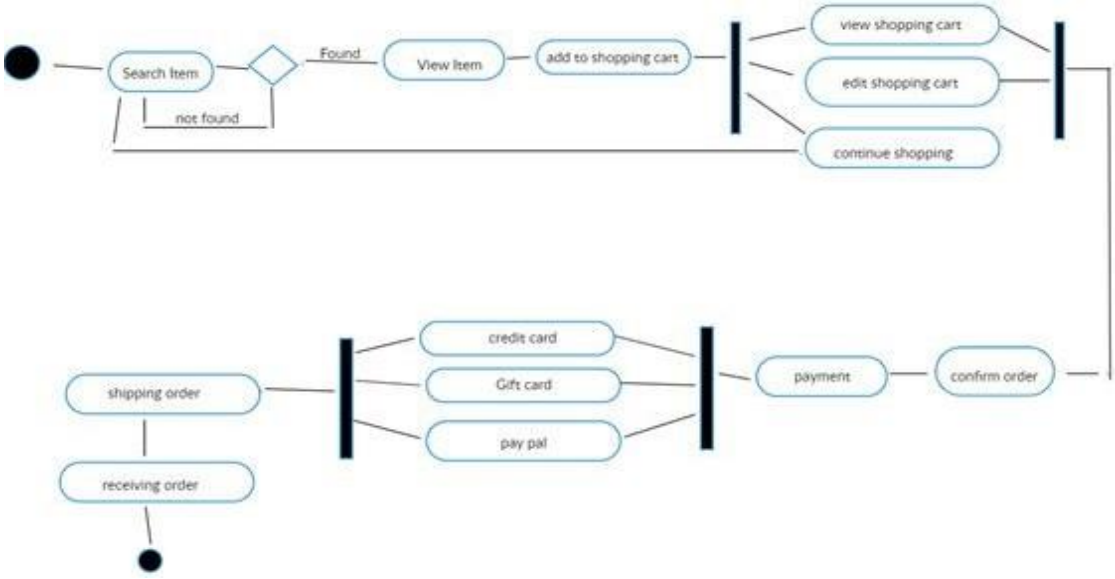
Implementation and Testing

UrDoorBoT

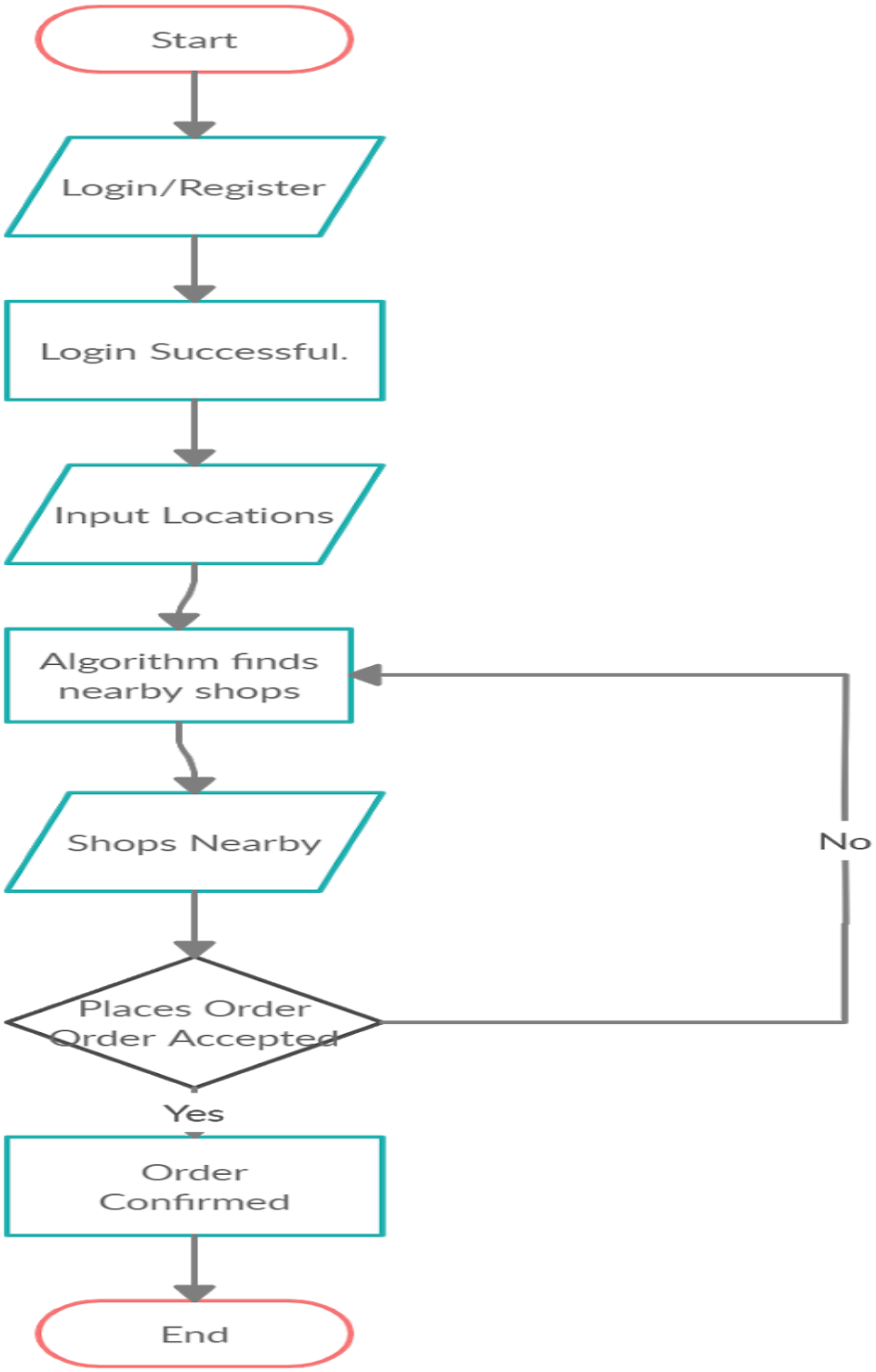
1.Implementation

1.1 Block Diagram:





1.2 Algorithm



1.3 Sample Code

Not Applicable

1.4 Module Wise Sample Screen Shots

AUTHORIZATION MODULE

```
// Register Route

router.get('/userreg', (req, res) => {
    res.render('register')
})


router.post("/userreg", async(req, res) => {
    try{
        var newUser=new User({name:req.body.name,username:req.body.username});
        // waiting for this to register
        const registeredUser = await User.register(newUser,req.body.password)
        // res.status(201).send(newUser)
        res.redirect('/')
        // logging just to ensure the user is saved

    }
    catch(e){
        //throwing errors
        console.log(e)
    }

})
```

In the register route first we have rendered the register.ejs file(front-end of register form) when the user requests a GET command. After filling the registration form when the user hits the submit button the POST route gets executed and it fetches the credentials entered by the user using the req (request) object.

The fetched credential is then passed into the User Schema from which we are creating a newUser and saving it to the database(in this case MongoDB). Since

the register function takes time we have used async function so that the flow of the code doesn’t get disturbed. After the register is saved the user is then redirected to the main landing page of the website and if during

saving the user's credential any error occurs we catch it and log it on our server using console.log(e).

```
// LOGIN ROUTE

router.get("/login", (req, res) => {
    res.render("login")
})
router.post('/login',passport.authenticate('local', {
      failureRedirect: '/login',
    }),(req, res)=>{
        res.redirect('/')

  });
```

For authorization we have used passport.js library which provides us various methods of authentication. In this login route we have first rendered the login.ejs file(front-end page of login) on GET request. After the user is done filling the form and hits the login button ,POST request gets executed and since we have used passport.js for authentication we get a middleware in built which we are using over here, passport.authenticate() authenticates the fetched data and checks whether the user is already registered or not in the database. If it fails to find the user it redirects the user to the login page again and if it finds the user then the user will get redirected to the landing page of the website.

```
//Logout Route


router.get("/logout",function(req,res){
    req.logout();
    res.redirect("/")
});
```

For logout we have again used the inbuilt functionality of passport.js. When the user hits the logout route req.logout() ends the user session by deleting it and redirects it to the home page.

ITEMS MODULE

```
//Index Route
router.get("/",function(req,res){
    //get campgrounds from DB
    Owner.find({category:"Grocery"},function(err,allgrocery){
        if(err){
            console.log(err);
        }else{
            res.render("grocerypage",{Owner:allgrocery});
        }
    });

});
```

When the user selects the type of shop in the main page according to their selection we display the shops (in this case we have assumed that the user has selected the grocery shop). When the user selects a particular type of shop GET request is fired and we make a request to our database to find all the shops with category:"Grocery" (according to the user's selection). We then pass all the results we get from the database to the corresponding ejs template (in this case grocery page).

```
// SHOW - show more info
router.get("/:id",function(req,res){

    Owner.findById(req.params.id,function(err,foundgrocery){
        if(err){
            console.log(err);
        }else{
            res.render("grocerydesc",{Owner:foundgrocery});
            console.log(foundgrocery)
        }

    });

});
```

After all the results are displayed in that particular category if the user wants to select any particular shop and see its detail the user has to click on it , when the user clicks the shop we get that shop's ID which is provided by the database. Using that ID we then search for that particular shop and get all its data and then present it to the client.

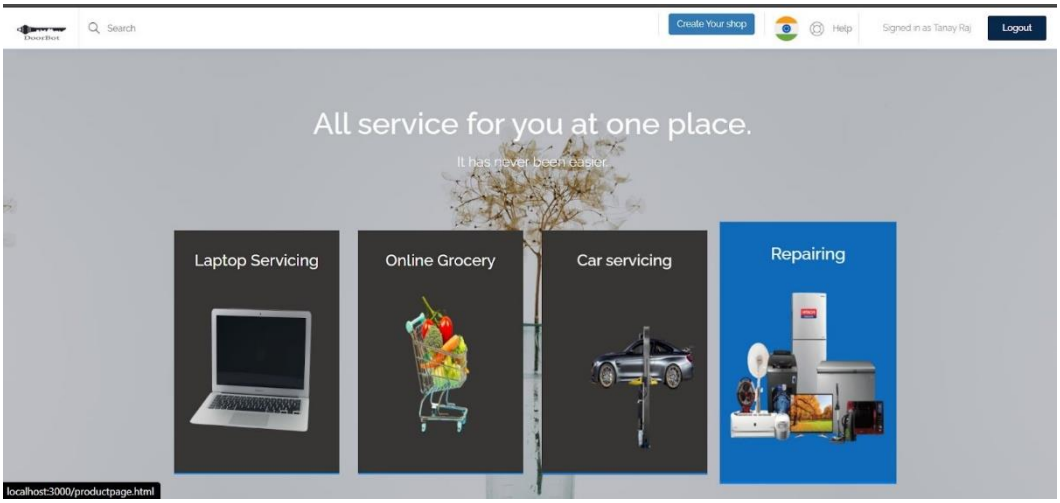
2. Testing

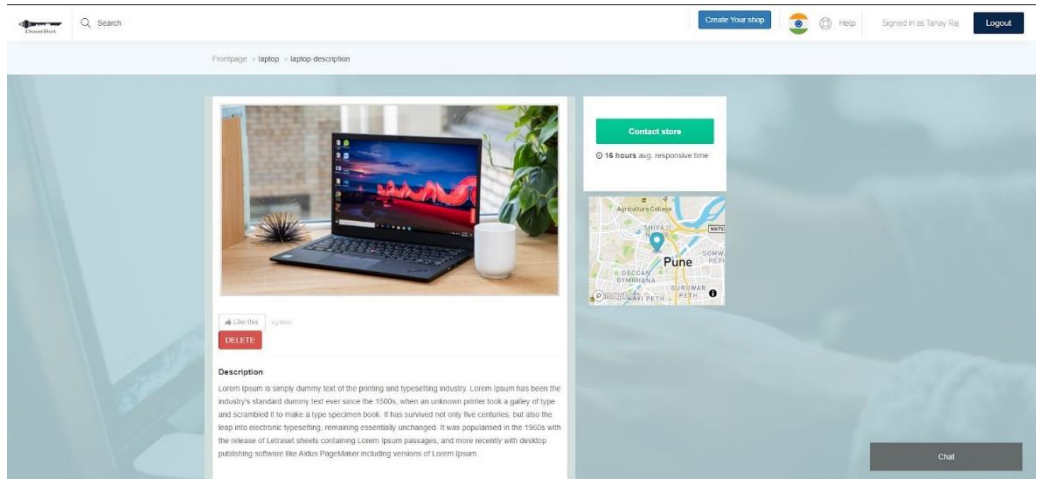
2.1 Testing Technique

We have used functional testing for our project. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

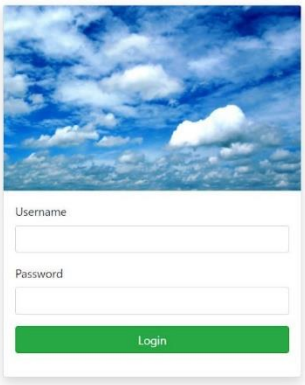
2.2 Test Case Generation

- TEST CASE-1 : It should GET all the Laptop stores.
- TEST CASE-2 : It should GET all the grocery stores.
- TEST CASE-3 : It should create a user.
- TEST CASE-4 : It should login a registered user.


Search
Create Your shop
Help
Signed in as Tanay Raj
Logout
All service for you at one place.
It has never been easier.
Laptop Servicing
Online Grocery
Car servicing
Repairing
localhost:3000/productpage.html


Search
Create Your shop
Help
Signed in as Tanay Raj
Logout
Frontpage > laptop > laptop description
Contact store
16 hours avg. responsive time
Pune
DELETE
Description
Chat


DoorBot
Login
Register


Username
Password
Login

2.3 Testing Tool

2.3.1 Details Of Testing Tools

For testing we have used the Mocha framework with Chai-http assertion library. Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.
Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

2.3.2 Configuration of Testing Tool

Test.js FILE

```
const expect = require('expect');
const request = require('supertest');
let chai = require('chai');
let chaiHttp = require('chai-http')
const User = require('../model/Users')
// let server = require("../server")

//Assertion Style
chai.should();

chai.use(chaiHttp);
```

We first require all the important libraries, frameworks and files we need for testing.

FOR TEST CASE-1 : It should GET all the Laptop stores

```
//GET route
describe('GET /laptop/',()=>{
    it("it should GET all the laptop stores",(done)=>{
        chai.request('http://localhost:3000')
            .get('/laptop/')
            .end((err,response)=>{
                response.should.have.status(200)
            done();
            })
    })

})
```

We first describe the event and then we hit the url http://localhost:3000 (local server). We then perform a GET request on the /laptop/ route and then we check the status code if it is 200 or not(The HTTP 200 OK success status response code indicates that the request has succeeded.)

FOR TEST CASE-2 : It should GET all the grocery stores.

```
describe('GET /grocery/',()=>{
    it("it should GET all the grocery stores",(done)=>{
        chai.request('http://localhost:3000')
            .get('/grocery/')
            .end((err,response)=>{
                response.should.have.status(200)
            done();
            })
    })

})
```

We first describe the event and then we hit the url http://localhost:3000 (local server). We then perform a GET request on the /grocery/ route and then we check the status code if it is 200 or not(The HTTP 200 OK success status response code indicates that the request has succeeded.)

FOR TEST CASE-3 : It should create a user.

```
describe('post /userreg/',()=>{
    it('should create a user', (done) => {
        var name = "DoorBot112";
        var password = "DoorBotIstheBest";
        var username = "DG123T"

        chai.request("http://localhost:3000")
          .post('/userreg')
          .send({name, password,username})
          .end((err,response)=>{
            response.should.have.status(200)
        done();

          });
      })

})
```

For checking the registration functionality, we create a new user object and pass it to the http://localhost:3000/userreg/ route as a body. And then again we check for the status code 200. If it is 200 then the response is OK otherwise some error must be there.

FOR TEST CASE-4 : It should login a registered user.

```
describe('post /login/',()=>{
    it('should login a user', (done) => {
        var password = "DoorBotIstheBest";
        var username = "DG123T"

        chai.request("http://localhost:3000")
          .post('/login')
          .send({username,password})
          .end((err,response)=>{
            response.should.have.status(200)
        done();

          });
      })

})
```

We pass the same user object because it has already been registered in the login route and check for the status code 200.

TEST CASES RESULTS

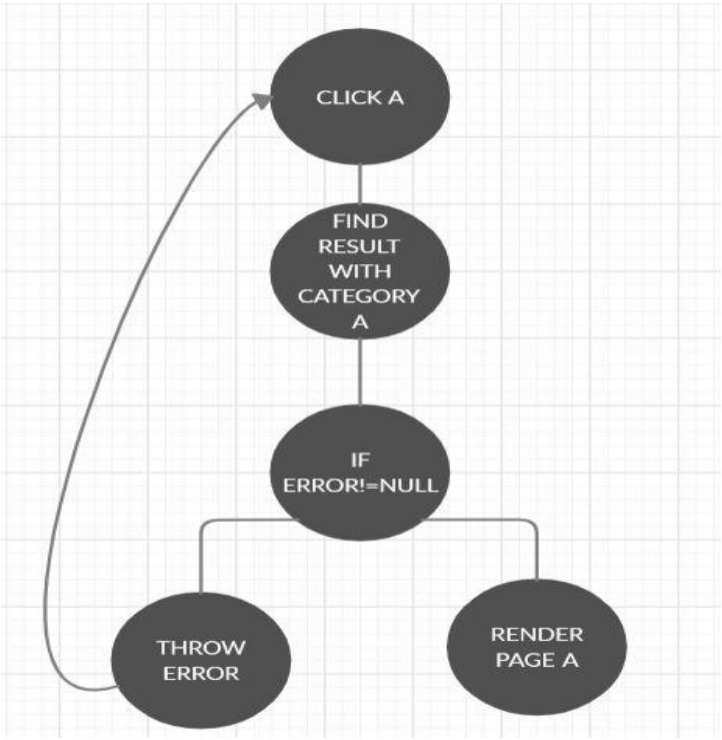
```
  API
    GET /laptop/
      √ it should GET all the laptop stores (315ms)
    GET /grocery/
      √ it should GET all the grocery stores (41ms)
    post /userreg/
(node:19612) [DEP0066] DeprecationWarning: OutgoingMessage.prototype._headers is deprecated
(Use `node --trace-deprecation ...` to show where the warning was created)
      √ should create a user (1322ms)
    post /login/
      √ should login a user (1184ms)


  4 passing (3s)
```

As we can see all the 4 test cases have successfully passed which

shows that all the functionalities are working perfectly.

2.3.3 Graph

```
1.  Owner.find({category:"A"},function(err,allA){
2.      if(err){
3.          console.log(err);
4.      }else{
5.      res.render("page A",{Owner:allA}); 6.
        }
7.  });
```



Cyclomatic complexity = E - N
+ 2*P where,
E = number of edges in the flow graph.
N = number of nodes in the flow graph.
P = number of nodes that have exit

points Cyclomatic complexity = 5 –

$5 + 2*1 = 1$

THANK YOU.