

DeliveRx

Online Pharmacy Application

Final Project Report

INFSCI 2560 – Network and Web Data Technologies

DeliveRx



University of Pittsburgh, School of Computing & Information



Submitted by:

Name	Student ID	Pitt Email Address
David Ball	4396643	dab304@pitt.edu
Piu Mallick	4374215	pim16@pitt.edu
Chuhan Xu	4373189	chx37@pitt.edu

Table of Content

Introduction	- 3 -
Technologies.....	- 3 -
Application Details.....	- 3 -
Users:	- 3 -
Features and Functionalities	- 3 -
Assumptions.....	- 5 -
Application Flow & Screenshots	- 5 -
Home Page	- 5 -
Login Page	- 6 -
Registration Page.....	- 7 -
Customer Landing Page / Customer Dashboard.....	- 8 -
Customer Placing Order	- 8 -
Add Cart / Clear Cart	- 9 -
Order History Page.....	- 9 -
Customer Profile Page.....	- 10 -
Staff Landing Page / Staff Dashboard.....	- 10 -
Medicines Page.....	- 10 -
Suppliers Page	- 11 -
Design and Accessibility.....	- 11 -
Front-End	- 15 -
Back-End	- 15 -
Database Design	- 17 -
Authentication & Authorization	- 19 -
User authentication flow:.....	- 19 -
Error Handling	- 21 -
Future Improvements	- 23 -
Code Base.....	- 23 -
Project Responsibilities	- 23 -
References.....	- 23 -

Introduction

This project represents a full-stack implementation of an e-commerce website which handles online delivery of Over-the-Counter medicine to customers within the Pittsburgh region. While delivery of prescription drugs is a well-established service, quick delivery of basic yet essential healthcare products is a mostly untapped market. **The demand for such a service is only heightened by the COVID-19 pandemic and current need for social distancing.**

Technologies

- **Front-End:** HTML5 / CSS3, JavaScript
- **Back-end:** NodeJS Express
- **Database:** PostgreSQL



Application Details

The online store contains medicines that can be categorized by types. The medicines can be browsed and searched by customers on the website and can be added to the cart to make a purchase.

Users:

The application **has 2 kinds of users/actors**. They are stated as follows:

- **Customer** – a person who can purchase medicines.
- **Staff** – a person who can manage the inventory details and view the order details of the customers.

The application has a login page, generic for all types of users, viz. **Customer** and **Staff**. There is a **role-based authentication process**, which means all the users/actors can log-in using the same **Login page**. However, they will be directed to a different portal after log-in, according to their access/roles.

Features and Functionalities

Guest Features

- Any unregistered user can visit the website and explore – get to know more about the online store and search medicines.
- If the user intends to buy medicine(s), they he/she has to login to the system and continue shopping.

Login / Registration

- All the users can login to the system if they have valid **username** (**Email Address** in this case) and **Password**. That means, they need to '**Sign Up**' before they can buy medicines.
- If they are not registered in the system, they need to do the registration by providing few details, like **First Name**, **Last Name**, **Email Address**, **Phone Number**, **Address**, **Gender**, **Date of Birth** and **Password**.
- Once done, they can login using the same **Email Address** and **Password** they have used for registration.
- After log-in, they can view a '**Welcome Page**' with features respective to their roles.

Customer Specific Features

- A customer can visit the website, explore various medicines, and if he / she decides to purchase medicines, he/she can register (if new) and login to the system and add medicines to the shopping cart and make a purchase.
- Once logged in, the customer is greeted with a '**Welcome Message**'.
- The customer would be able to view the features available to him / her, viz:
 - View his/her personal information (in the '**Profile**' page).
 - Add the desired medicines in the shopping cart, and then check out. For each medicine item added in the cart, a message will be displayed showing the confirmation of that item added to the cart.
 - After checkout, a message will be displayed on the screen with order and delivery details.
 - He / She can view his / her order details in the '**Order History**' page.
 - The customer can '**Log Out**' when desired.

Staff Specific Features

- A staff can log in using the '**Log In**' portal.
- Once logged in, the staff is redirected to '**Staff Dashboard**' and greeted with a '**Welcome Message**'.
- The staff can '**Log Out**' when desired.
- He / She can navigate between the pages and return to the main dashboard, when desired.
- He / She would be able to view the features available to him / her, viz.:
 - **Access to Medicine List**
 - View Medicines
 - Add Medicines
 - Edit / Update Medicine details

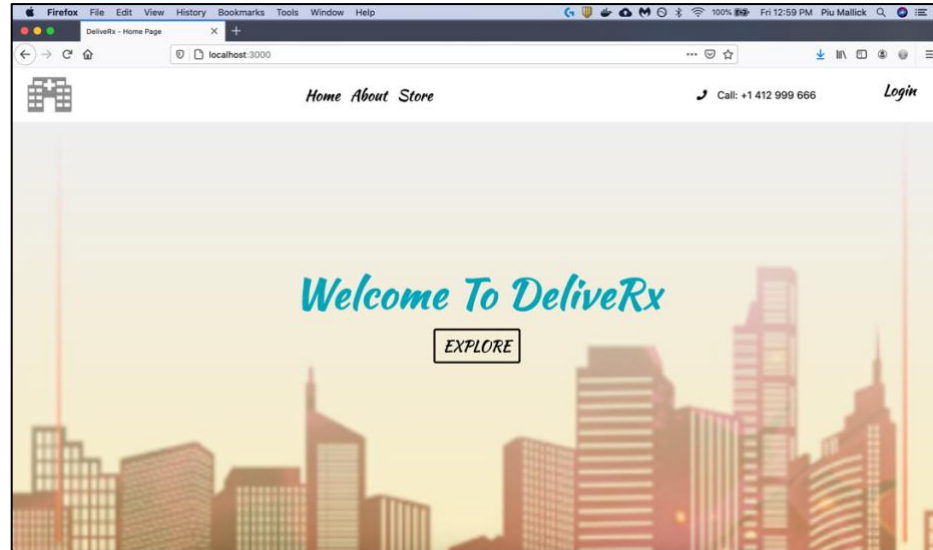
- Delete Medicines
- **Access to Supplier List**
 - View Suppliers
 - Add Suppliers
 - Edit / Update Supplier Details
 - Delete Suppliers

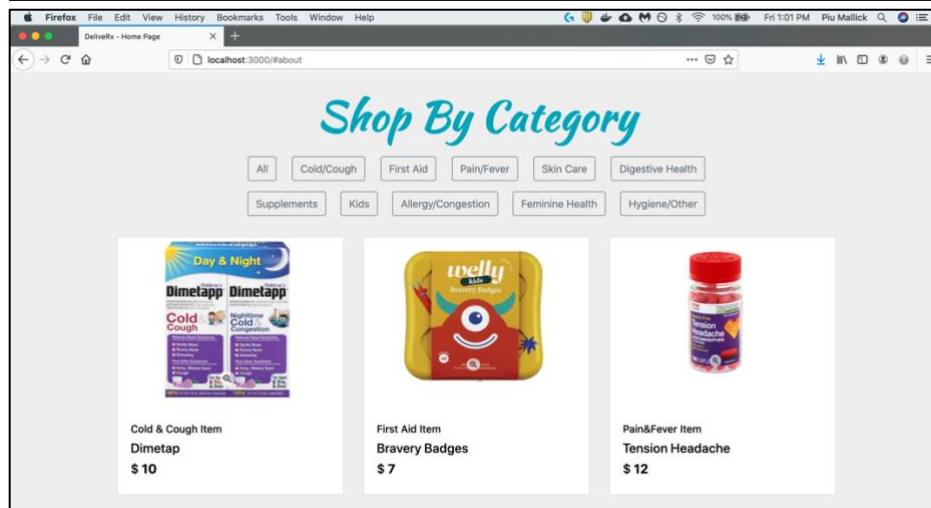
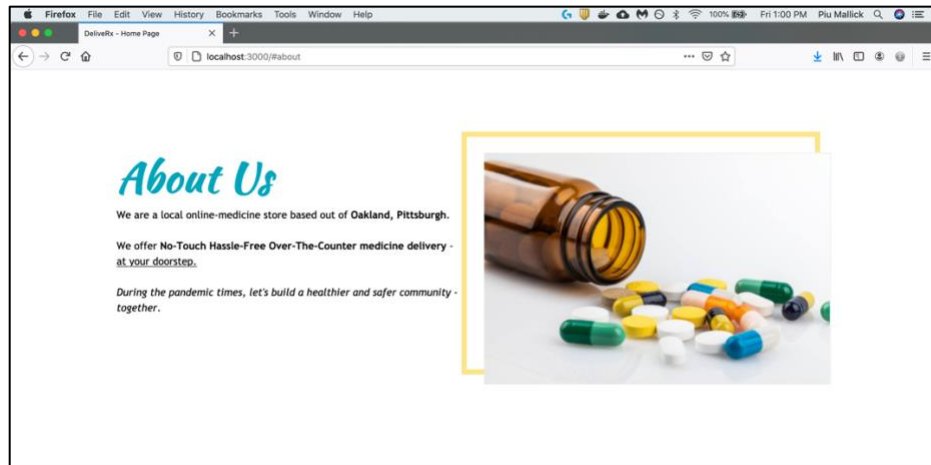
Assumptions

- The staff has already been created from the backend. He / She has the admin responsibilities of the online store.
- The payment gateway functionality has not been implemented after the customer checkout. The functionality is limited to when the customer gets a message for successful order placement and delivery.

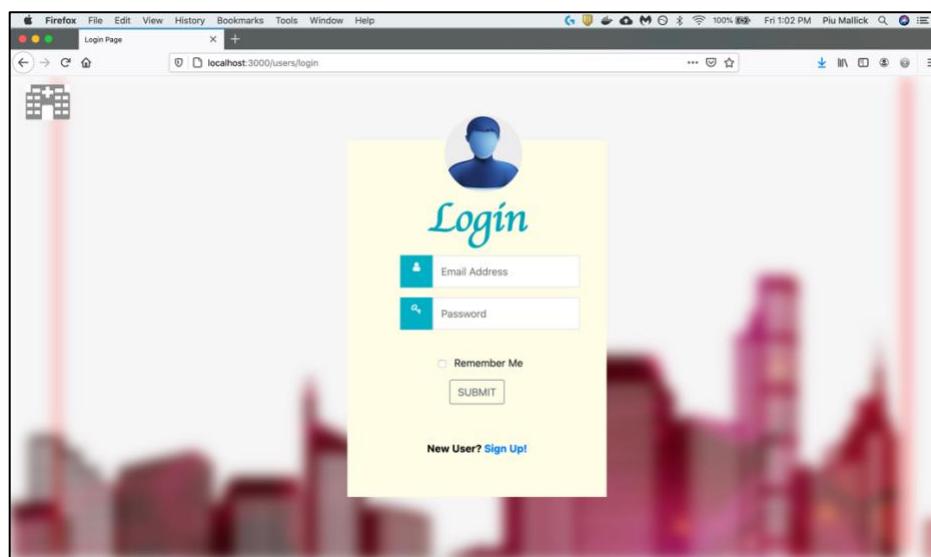
Application Flow & Screenshots

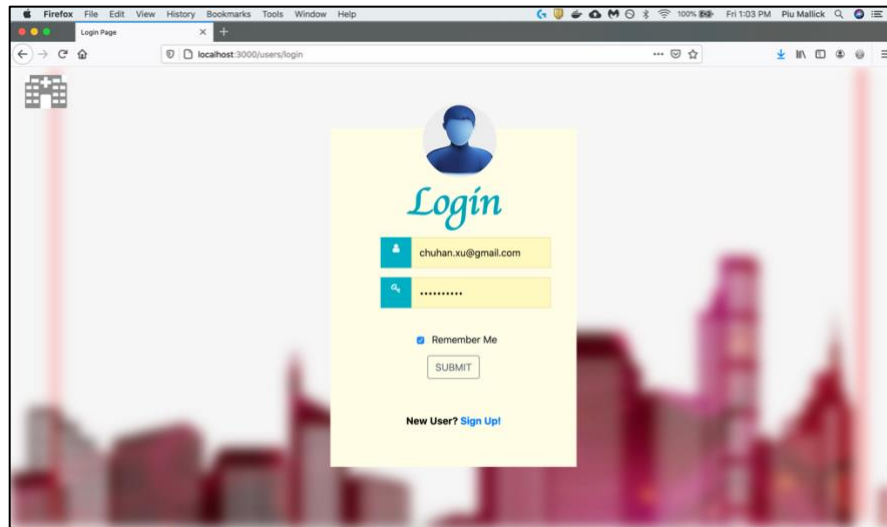
Home Page





Login Page

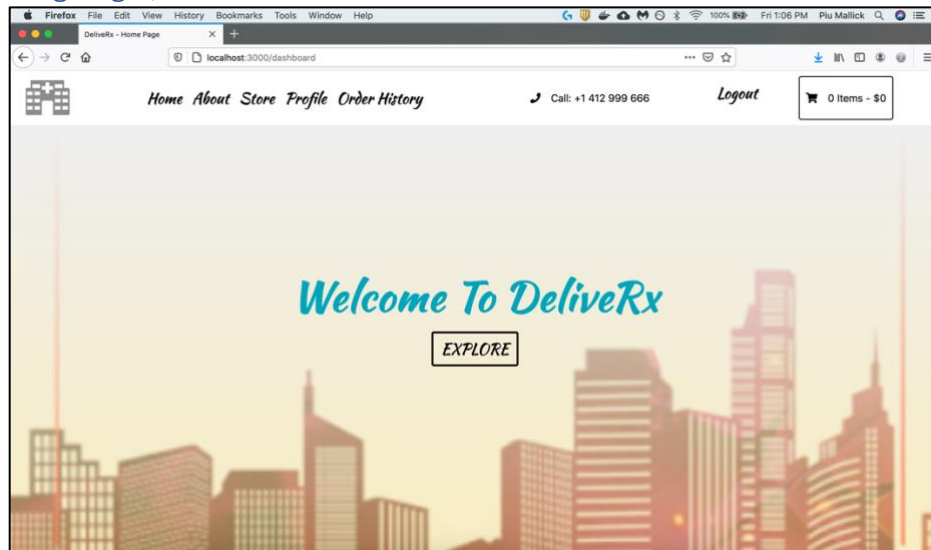




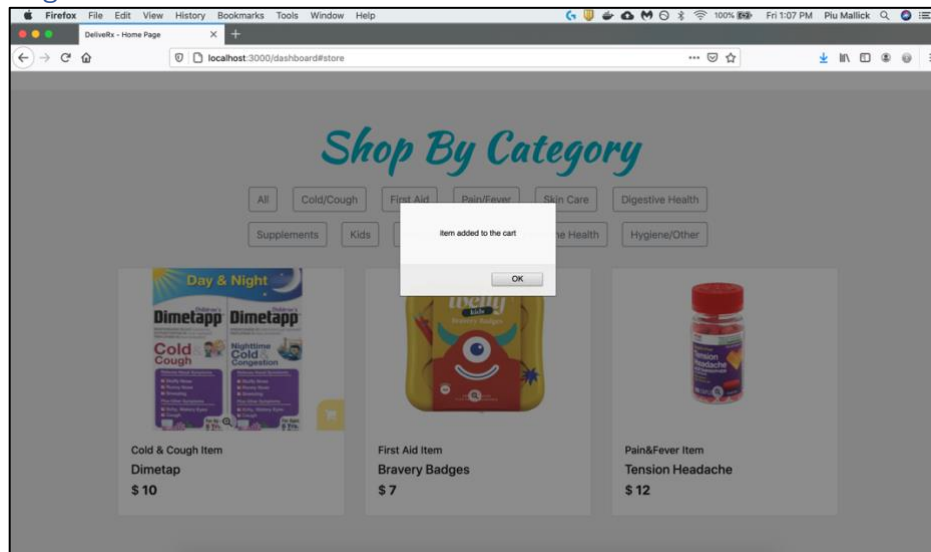
Registration Page

A screenshot of a web browser displaying a registration page. The page features a yellow central card with the word "Register" in a stylized font. Below the word are several input fields: "First Name", "Last Name", "Email Address", "Phone Number", "Address", "Gender", "Date of Birth", "Password", and "Confirm Password". A "CONTINUE" button is located at the bottom of the card. Below the card, it says "Already have an account? Sign In!" with a blue link. The browser's address bar shows "localhost:3000/users/register".

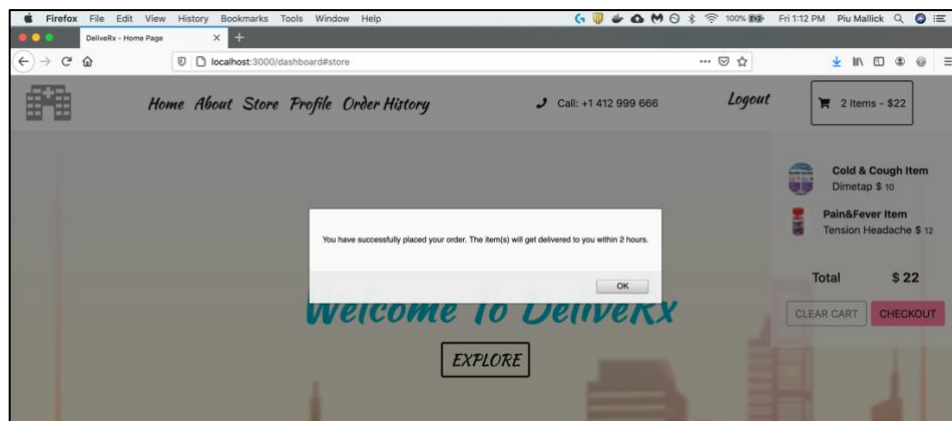
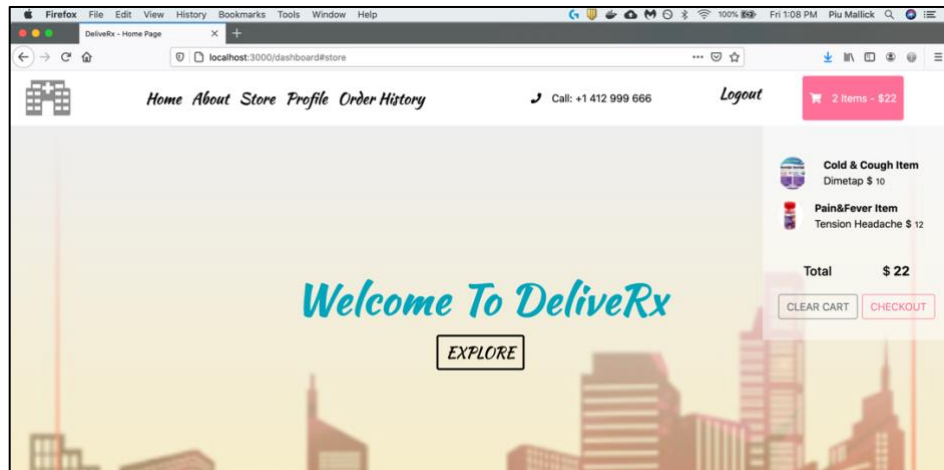
Customer Landing Page / Customer Dashboard



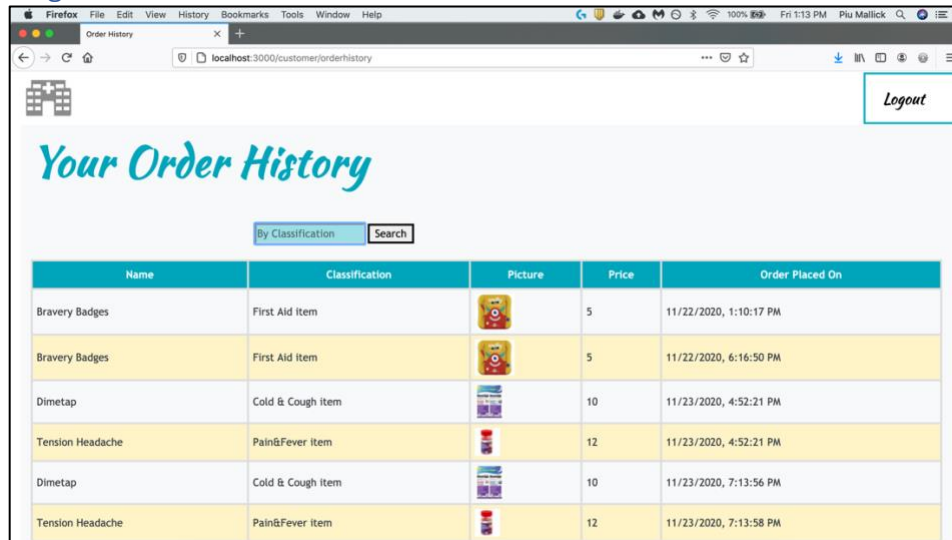
Customer Placing Order



Add Cart / Clear Cart



Order History Page



Customer Profile Page

Your Profile

[Logout](#)

Name:	Chuhan Xu
Email:	chuhan.xu@gmail.com
Phone #:	4129987654
Address:	Pittsburgh
Date of Birth:	9/15/1998, 12:00:00 AM

[Return to Main Page](#)

Staff Landing Page / Staff Dashboard

Staff Dashboard

Welcome ! Have a nice day at work!

Medicine List

You can do the following activities:

- Add Medicines
- Update Medicines
- Delete Medicines

Supplier List

You can do the following activities:

- Add Suppliers
- Update Suppliers
- Delete Suppliers

Medicines Page

Add / Modify / Delete Medicine(s)

Medicine List

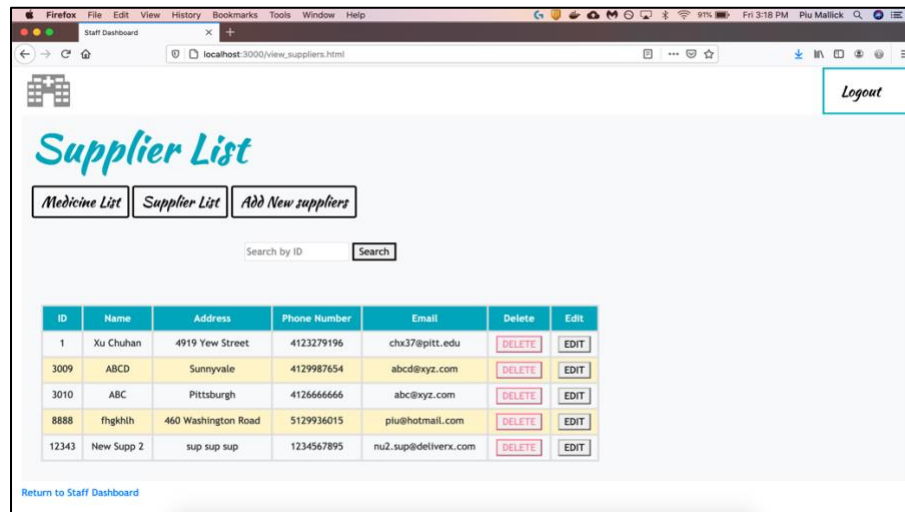
[Medicine List](#) [Supplier List](#) [Add New Medicine](#)

Search by Id

ID	Name	Price	Classification	Picture	Delete	Edit
1013	Loratadine D	9.00	Allergy/Congestion		Delete	Edit
1015	Pseudoephedrine 30 mg	2.00	Allergy/Congestion		Delete	Edit
1016	Pseudoephedrine 12HR	5.00	Allergy/Congestion		Delete	Edit
1017	Oxymetazoline Nasal Spray	3.00	Allergy/Congestion		Delete	Edit

Suppliers Page

Add / Modify / Delete Supplier(s)



Design and Accessibility

Efforts have been put in to make sure that the website is friendly to all kinds of users, independent of their physical or mental abilities.

- **CSS design choices**

Font Awesome has been used for designing the web pages. It is the most popular way to add font to the websites. **Font Awesome icons** are created using **scalable vectors**, so that we can use high quality icons that work well on any screen size.

Sample Code Snippet (used in 'all.css' file)

```
.fa-font-awesome:before {  
  content: "\f286"; }  
  
.fa-font-awesome-alt:before {  
  content: "\f3a3"; }
```

Also, **contrasting colors** (light background with dark fonts and images) have been chosen for better readability and understandability.

- **Concise Page Titles**

Appropriate web page titles have been given across all the pages so that they concisely describe the content of the pages and each page can be uniquely identified from one another.

Sample Code Screenshots

Home Page

```
<!-- font awesome -->
<link rel="stylesheet" href="css/all.css">
<title>DeliveRx - Home Page</title>
<style>
</style>
</head>
```

Customer Profile Page

```
<link
  rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
  integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpsL8n8PARn9HuZ0nIxN0hoP+VmmDGMN5t9U"
  crossorigin="anonymous"
/>
<title>Customer Profile Page</title>
```

- **Responsive design**

Responsive design is a set of techniques for structuring **HTML** and **CSS** so that the web pages are readable on multiple devices.

The responsive design has been achieved through use of **Bootstrap** and **Viewport**, making the website a pleasure to browse and enjoy on any kind of device.

Code Snippet:

```
view_suppliers.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <title>Staff Dashboard</title>
5 <meta name="viewport" content="width=device-width, initial-scale=1"/>
6 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
7 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
8 <!-- Bootstrap CSS -->
9 <link
10   rel="stylesheet"
11   href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
12   integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpsL8n8PARn9HuZ0nIxN0hoP+VmmDGMN5t9UJ0Z"
13   crossorigin="anonymous"
14 />
```

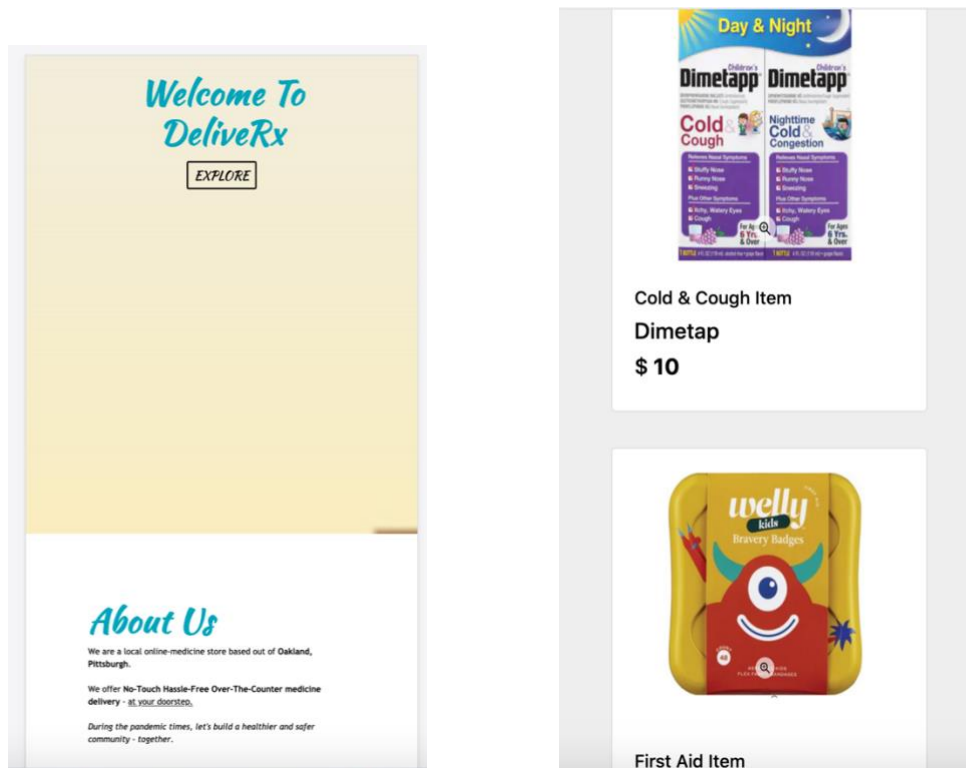
The above code snippet has been used in all the pages to make the web application as much responsive as possible.

Various **div-classes** like “**container-fluid**”, “**col-md-8**”, “**col-sm**”, etc. have been used to achieve different page layouts as per defined by the **Bootstrap library**.

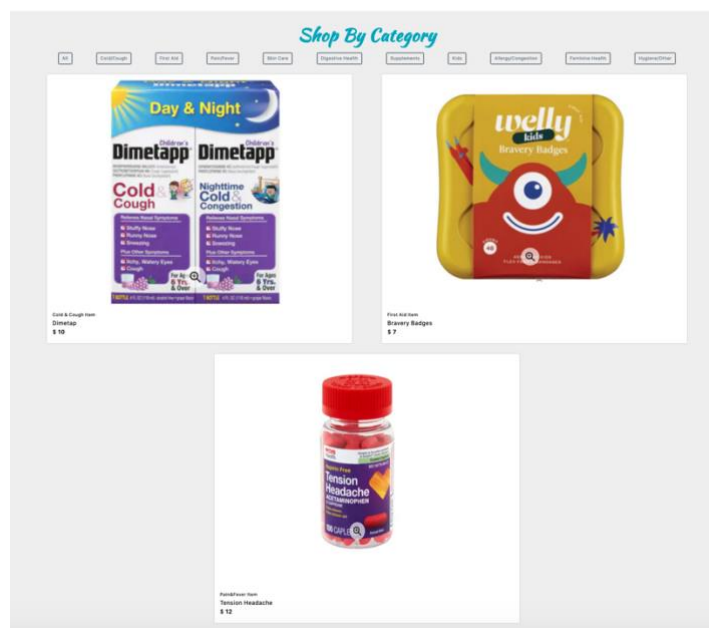
(Multi-column layout that stacks the columns vertically as screen size narrows)

Screenshot for iPhone X/XS iOS 12 – 375 x 812

(Showing **Responsiveness** of the web pages)



Screenshot for iPad – 768 x 1024 (Showing **Responsiveness** of the web pages)



▪ Responsive Images

We have made the images responsive as well by setting the ‘**max-width**’ attribute to **100%**.

Sample Code Screenshot (“bootstrap.min.css”)

```
legend {  
  display: block;  
  width: 100%;  
  max-width: 100%;  
  padding: 0;  
  margin-bottom: .5rem;  
  font-size: 1.5rem;  
  line-height: inherit;  
  color: inherit;  
  white-space: normal  
}
```

▪ Accessibility concerns

The **Web Content Accessibility Guidelines (WCAG)** is organized around four principles, and they go by the acronym **POUR**. We have tried to implement these principles in our web application.

- **Perceivable (P)** → As our web application is pretty heavy on images, it has been made perceivable by providing text alternatives for the images (i.e. non-text content).
- **Operable (O)** → The web application has been made operable by making the navigation possible through keyboard, especially in the Login and Registration pages.
- **Understandable (U)** → The web application is understandable in nature – the texts are readable and understandable for the users, making the use of clearest and simplest language possible. The content appears and operated in predicted in predictable ways.
- **Robust (R)** → The web page content can be consumed by a wide variety of user agents (browsers) and the content is compatible with current and future tools. The markup can be reliably interpreted.

▪ Limitations

Since some of the pages contain tabular data, it is difficult to fit all of it into the small screen, especially in the smart phone screens. This limitation can be handled in a different way by modifying the display of data in tiles-format when the screen size shrinks. We plan to add this feature in future.

Mobile First Design

The home page and the customer pages meet the requirements of the 'Mobile First Design' approach, which means the pages fit well in the small screen. We are working on making the staff pages to fit in the small screen as well, by making appropriate design changes.

Front-End

- **HTML, JS code examples (for Shopping Cart)**

The function shown below acquires information of the specific medicine item from item card and display it into shopping cart.

```
//add items to the cart
(function(){
  //make sure we click the shopping-cart,if you click yellow span it won't work
  const carBtn = document.querySelectorAll(".store-item-icon");
  carBtn.forEach(btn=> {
    btn.addEventListener("click",event=>{
      //target element
      // console.log(event.target.parentElement.classList.contains("store-item-icon"));
      // img-container/store-item-icon/fas fa-shopping-cart
      if(event.target.parentElement.classList.contains("store-item-icon")){
        //got path of img
        let fullPath=event.target.parentElement.previousElementSibling.src;
        let pos = fullPath.indexOf("img")+3;
        let partPath=fullPath.slice(pos);
        // item(img: "img-cart/firstaid.jpg", classification: "First Aid item", name: "Bravery Badges", price: "5")
        const item={};
        item.img="img-cart${partPath}";

        let classification = event.target.parentElement.parentElement.nextElementSibling.children[0].children[0].textContent;
        item.classification = classification

        let name = event.target.parentElement.parentElement.nextElementSibling.children[0].children[1].textContent;//card-body
        item.name = name

        let price=event.target.parentElement.parentElement.nextElementSibling.children[0].children[2].textContent;
        let finalPrice = price.slice(1).trim() //price = $ 5 we need to get rid of $
        item.price=finalPrice
      }
    })
  })
})
```

Back-End

There are three main steps for **back-end** about the **insert**, **delete** and **update** functions.

Example about delete function

- **Step1: Build Router**

```
function deleteRowById(id) {
  fetch('./staff/view_medicines/delete/' + id, {
    method: 'DELETE'
  })
  .then(response => response.json())
  //just return true or false
  .then(data => {
    console.log(data)
    if (data.success) {
      location.reload();
    }
  });
}
```

- Step2: Initialize database service instance

```
// delete
router.delete('/view_medicines/delete/:id', (request, response) => {
  console.log(request.params)
  const { id } = request.params;
  const db = dbService.getDbServiceInstance();

  const result = db.deleteRowById(id);

  result//true or false
  .then(data => response.json({success : data}))
  .catch(err => console.log(err));
});
```

- Step3: Specify SQL syntax according to different operations

```
// delete for medicine
async deleteRowById(medicine_id) {
  try {
    let id = parseInt(medicine_id, 10); //10 is base
    const response = await new Promise((resolve, reject) => {
      const query = "DELETE FROM medicine_category_price WHERE medicine_id = $1 ;"

      pool.query(query, [medicine_id], (err, result) => {
        if (err) reject(new Error(err.message));
        resolve(result.rowCount); //rowCount 是要被删除的那一行
      })
    });
    // console.log(response);
    return response === 1 ? true : false;
  } catch (error) {
    console.log(error);
    return false;
  }
}
```


Database Design

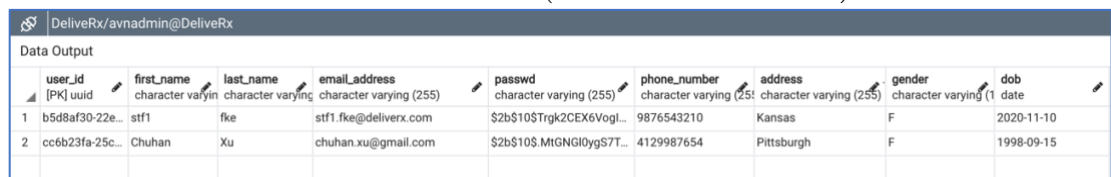
PostgreSQL has been used as the database for this web application.

Some of the main business rule include:

- User can be a customer or a staff.
- One customer can place many orders.
- One order is only from one customer.
- One order can contain many items.
- One category has many medicines.

The following tables have been used:

- **USERS** → This table contains the user details (customer and staff details).



	user_id [PK] uuid	first_name character varying	last_name character varying	email_address character varying (255)	passwd character varying (255)	phone_number character varying (255)	address character varying (255)	gender character varying (1)	dob date
1	b5d8af30-22e...	stf1	fke	stf1.fke@deliverx.com	\$2b\$10\$Trgk2CEX6Vogl...	9876543210	Kansas	F	2020-11-10
2	cc6b23fa-25c...	Chuhan	Xu	chuhan.xu@gmail.com	\$2b\$10\$.MtNGl0yg\$7T...	4129987654	Pittsburgh	F	1998-09-15

The notable columns in the **USERS** table are:

user_id, first_name, last_name, email_address, passwd (password), phone_number, address, gender, dob (date of birth)

The datatype of user_id is **uuid**, which stores **Universally Unique Identifiers (UUID)** defined by **RFC 4122** and other related standards. A **UUID** value is 128-bit quantity generated by an algorithm that make it unique in the known universe using the same algorithm. **UUID** is a sequence of **32 digits of hexadecimal digits**, represented in groups and separated by hyphens. Because of its uniqueness feature, we often find **UUID** in **distributed systems** because it guarantees a better uniqueness than the SERIAL data type which generates only unique values with a single database. Keeping this unique feature in mind, we have also tried to put in the use of **UUID** so that it could be useful when we go for future enhancements in our web application.

The password (**passwd**) field is **hash encrypted**, as shown in the above screenshot. It is better way of encryption and adds another level of security.

- **ROLE** → This table contains the role details (role id and name – whether a staff or a customer).

DeliveRx/avnadmin@DeliveRx		
Data Output		
	role_id [PK] integer	role_desc character varying (10)
1		2 STAFF
2		3 CUSTOMER

- **ROLE_MAPPING** → This table maps the user to its corresponding role.

DeliveRx/avnadmin@DeliveRx							
Data Output							
	user_id [PK] uuid	role_id [PK] integer	isactive character varying (1)	create_date date	end_date date	modified_date date	
1	b5d8af30-22e...	2	Y	2020-11-10	[null]	2020-11-10	
2	cc6b23fa-25c...	3	Y	2020-11-10	[null]	2020-11-10	

This table contains the role creation details – while customer registration, a mapping happens with the **user_id** and **role_id**, from **users** and **role** table respectively.

- **MEDICINE_CATEGORY_PRICE** → This table contains the medicine details – name, category, price, etc.

DeliveRx/avnadmin@DeliveRx					
Data Output					
	medicine_id integer	medicine_name character varying (255)	round numeric	category_name character varying (20)	picture text
1	1013	Loratadine D	9.00	Allergy/Congestion	LoratadineD.jpeg
2	1004	Naxproxen 200mg	2.50	Pain/Fever	Naproxen_200mg.jpg
3	1075	Band Aids Assorted	3.00	Hygiene/Other	Band_Aids_Assorted.jpeg
4	1028	Advil Cold & Sinus	10.50	Cold/ Cough	Advil_Cold_&_Sinus.jpeg
5	1041	Probiotic Blend	7.50	Digestive Health	Probiotic_Blend.jpg

- **ORDERS** → This table contains the order details placed by the customer.

DeliveRx/avnadmin@DeliveRx						
Data Output						
	order_name character varying (50)	order_classification character varying (50)	picture character varying (50)	order_price double precision	dateadded character varying (50)	
1	Tension Headache	Pain&Fever item	img-cart/painfever.jpg		5 11/14/2020, 12:14:39 PM	
2	Bravery Badges	First Aid item	img-cart/firstaid.jpg		5 11/14/2020, 12:14:39 PM	
3	Bravery Badges	First Aid item	img-cart/firstaid.jpg		5 11/14/2020, 12:15:55 PM	
4	Bravery Badges	First Aid item	img-cart/firstaid.jpg		5 11/14/2020, 12:16:59 PM	
5	Bravery Badges	First Aid item	img-cart/firstaid.jpg		5 11/14/2020, 12:21:52 PM	

Please note: For simplicity and time crunch, we have not followed all the **ACID properties** that should have been maintained in the **RDBMS system**. Some of the data are replicated

and redundant, whereas, in other cases, the data manipulation has been done through front-end (using **JavaScript**).

Authentication & Authorization

User authentication flow

Existing user (customer in this case) can sign in using the **Login page** or new users can **Register**. The authorization and authentication code in the **Node Js backend** resides in the file **'auth.js'**. After registration, users can sign in using the credentials they have used to register. Once the user logs in, a session is created which is maintained via **Passport Js**, the authentication as well is done using Passport using a **LocalStrategy**.

During a customer registration, a user profile is created in the database, the password is hashed using **bcrypt with a salt** and stored. During login, the email address and password entered by the user is then matched against the records in the database, the entered password is hashed using the same mechanism as the stored one and then matched. If the email address is found and has matching password, then the user profile is stored in the session.

```
router.post('/login', function (req, res, next) {
  passport.authenticate('local', function (err, user, info) {
    if (err) { return next(err); }
    if (!user) { return res.render('./login', { error: "Credentials do not match." }); }
    req.login(user, function (err) {
      if (err) { return next(err); }
      if (req.body.remember) {
        // If user selects remember me,
        // Cookie expires after 30 days
        req.session.cookie.maxAge = 30 * 24 * 60 * 60 * 1000;
      } else {
        //Cookie expires at the end of the session
        req.session.cookie.expires = false;
      }
      return res.redirect(' ../dashboard');
    });
  })(req, res, next);
});

passport.use('local', new LocalStrategy({
  usernameField: 'email',
  passwordField: 'password',
  passReqToCallback: true
}, (req, username, password, done) => {
  loginAttempt();

  async function loginAttempt() {
    const client = await pool.connect()
    try {
      await client.query('BEGIN')
      let queryStr = // Query to find the user
      await JSON.stringify(client.query(queryStr, [username], function (err, result) {
        if (err) {
          return done(err)
        }
        if (result.rows[0] == null) {
          return done(null, false);
        } else {
          bcrypt.compare(password, result.rows[0].passwd, function (err, check) {
            if (err) {
              console.log('Error while checking password');
              return done();
            } else if (check) {
```

- Once the user is logged in, the pages she/he visits are authorized via the user profile from session. For example, the dashboard is different for customer and staff. So, a customer cannot visit the staff dashboard even if she/he knows the direct URL to the staff dashboard. For example, the login end point for both customer and staff is same (`router.post('/login')`). After login, both get redirected to dashboard (`res.redirect('./dashboard')`). However once in dashboard, the stored user profile is checked for proper role and based on the role the user is directed to the proper dashboard. Moreover, if the user is not authenticated or not in session, she/he will be redirected to the login page.

Error Handling

Unit Testing and **Peer testing** has been done manually for each and every functionality.

Examples:

- **Customer registration** has restrictions on **email address**, **phone number**, **date of birth** (**Customer needs to be at least 18 years**) and password matching fields. The interface rejects invalid entries and displays appropriate error messages.

The image displays three sequential screenshots of a web registration form titled 'Register'. Each form contains input fields for Name, Surname, Email, Phone Number, City, Gender, Date of Birth, Password, and Confirm Password. Below the fields is a 'CONTINUE' button and a link to 'Sign In!'. The first screenshot shows an error message 'The email address is not proper.' below the email field. The second screenshot shows an error message 'The phone number is not proper.' below the phone number field. The third screenshot shows an error message 'You must be 18 years of age or above.' below the date of birth field.

The above error messages have been handled in the following way (through 'auth.js')

```
router.post('/register', handlers.async function (req : Request<I>, ResBody, ReqBody, ReqQuery, , res : Response<ResBody> ) {
  let email = req.body.email;
  let phone_number = req.body.phone_number;

  if (!emailValidator.validate(email)) {
    res.render('view/register', options: {
      error: "The email address is not proper.",
      body: req.body
    });
    return;
  }
})
```

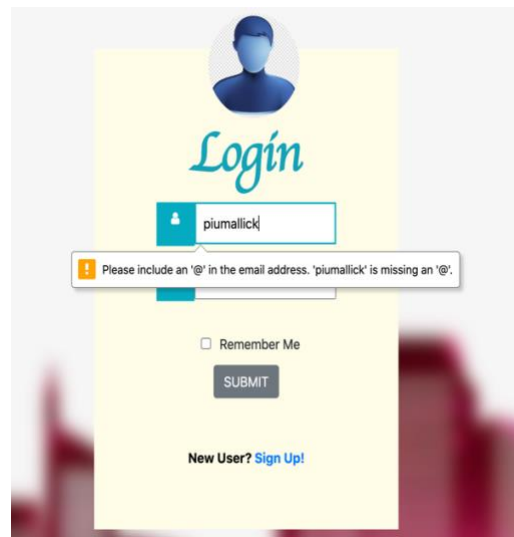
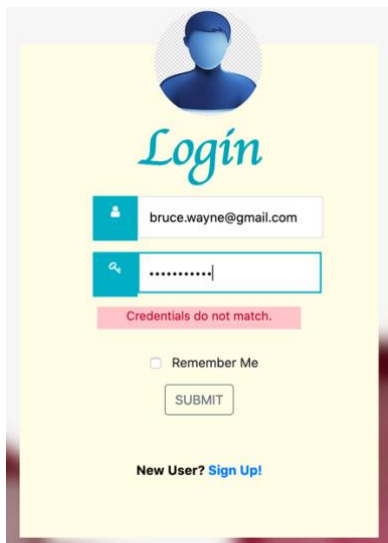
```

let dob = req.body.dob;
let age = moment().diff(moment(dob, "YYYY-MM-DD"), unitOfTime: 'years');
if (age < 18) {
  res.render( view: 'register', options: {
    error: "You must be 18 years of age or above.",
    body: req.body
  });
  return;
}

let phoneNumber = new PhoneNumber(phone_number, 'US');
if (!phoneNumber.isValid()) {
  res.render( view: 'register', options: {
    error: "The phone number is not proper.",
    body: req.body
  });
  return;
}

```

- Login error messages:



Flash messages have also been used for error handling in the console:

Sample Example (from 'auth.js')

```

router.get( path: '/register', handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> , next : NextFunction ) {
  res.render( view: 'register', options: {
    title: "Register",
    userData: req.user,
    messages: {danger: req.flash('danger'), warning: req.flash('warning'), success: req.flash('success')},
    error: null,
    body: null
  });
});

```

- **Addition of new medicine(s) and supplier(s) are rejected** if their corresponding data types are invalid. This has been taken care by **HTML5 Form Validation**.

Future Improvements

- Add more **error handling features** so as to make the web application more user-friendly.
- Add analytical reports in the ‘**Staff Dashboard**’ section so that the sales details can be viewed by them.
- Add more functionalities to make the web application more **robust** and **responsive**, and working towards the existing limitations.
- Make the database more robust by following the **ACID (Atomicity, Concurrency, Integrity & Durability)** principles that are followed in the Relational Database Management System (RDBMS).

Code Base

- **Github Link:** <https://github.com/piumallick/DeliveRx>
- **Database has been hosted in Cloud through Aiven:** <https://aiven.io/postgresql>

Project Responsibilities

Chuhan Xu	Piu Mallick	David Ball
Staff Functionalities	Authentication & Authorization	Customer Profile page
Customer Functionalities	CSS Page Styling	Error Handling
DeliveRx Home Page	Testing & Integration	Documentation

Database related tasks were taken up by individuals as per the module requirement. The database was hosted in cloud for ease and proper collaboration.

References

SL. No.	Functionality	Reference URL
1.	Home Page (CSS Template)	https://github.com/john-smilga/js-cart-setup/tree/master/js
2.	Login & Registration Functionality	<ul style="list-style-type: none"> ▪ https://medium.com/@timtamimi/getting-started-with-authentication-in-node-js-with-passport-and-postgresql-2219664b568c ▪ https://reallifeprogramming.com/node-authentication-with-passport-postgres-cf93e2d520e7 ▪ https://www.youtube.com/watch?v=vxu1RrR0vbw
3.	Font Styling	▪ https://fontawesome.com/