**DeliveRx**

**Online Pharmacy Application**

**Final Project Report**

**INFSCI 2560 – Network and Web Data Technologies**

|  |  |
| --- | --- |
| **A picture containing plate, drawing  Description automatically generated**  **A picture containing refrigerator, toy  Description automatically generated** | **A close up of a toy  Description automatically generated** |
|  |  |

**University of Pittsburgh, School of Computing & Information**

**Logo

Description automatically generated**

**Submitted by:**

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

1. **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**.

1. **Technologies**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **A close up of a sign  Description automatically generated** | **A close up of a sign  Description automatically generated** | **Icon  Description automatically generated** | **Text  Description automatically generated** |

1. **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.

1. **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.

1. **Features and Functionalities**
2. **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.

1. **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.

1. **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.

1. **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

1. **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.

1. **Application Flow & Screenshots**
2. **Home Page**

**A picture containing graphical user interface

Description automatically generated**

**Graphical user interface

Description automatically generated**

**Graphical user interface, website

Description automatically generated**

1. **Login Page**

**Graphical user interface, application, website

Description automatically generated**

**Graphical user interface, application, website

Description automatically generated**

1. **Registration Page**

**Graphical user interface, application, website

Description automatically generated**

1. **Customer Landing Page / Customer Dashboard**

**A picture containing text

Description automatically generated**

1. **Customer Placing Order**

**Graphical user interface, website

Description automatically generated**

1. **Add Cart / Clear Cart**

**A picture containing graphical user interface

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

1. **Order History Page**

**Graphical user interface, table

Description automatically generated**

1. **Customer Profile Page**

**Graphical user interface, application

Description automatically generated**

1. **Staff Landing Page / Staff Dashboard**

**Graphical user interface, application

Description automatically generated**

1. **Medicines Page**

**Add / Modify / Delete Medicine(s)**

**A picture containing graphical user interface

Description automatically generated**

1. **Suppliers Page**

**Add / Modify / Delete Supplier(s)**

**Graphical user interface

Description automatically generated**

1. **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)**

**Text

Description automatically generated**

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  Text  Description automatically generated | Customer Profile Page  Text  Description automatically generated |

* **Responsive design**

**Responsive design** is a set oof 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:**

Text

Description automatically generated

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**)**

|  |  |
| --- | --- |
| A picture containing application  Description automatically generated | Graphical user interface, application  Description automatically generated |

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

Graphical user interface, text

Description automatically generated

* **Responsive Images**

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

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

**Text

Description automatically generated**

* **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.

1. **Front-End**
   * HTML, JS code examples
   * Leave CSS for Design section?
2. **Back-End**
   * Node modules,
   * Router code
   * Other code examples?
   * What Express features did we utilize that we can discuss here? Server-side programming?
3. **Database Design**

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

Describe access procedures

Create table code?

Describe structure using below:

* 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).Graphical user interface, application

  Description automatically generated
* **ROLE** 🡪 This table contains the role details (role id and name – whether a staff or a customer).

Table

Description automatically generated

* **ROLE\_MAPPING** 🡪 This table maps the user to its corresponding role.

Table

Description automatically generated

* **MEDICINE \_CATEGORY\_ PRICE** 🡪 This table contains the medicine details – name, category, price, etc.

Graphical user interface, application

Description automatically generated

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

**Graphical user interface, application

Description automatically generated**

1. **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) {  
 let users = // creates user to store in the session  
 setUser(req, users); // sets user in session  
 return done(null, users);  
 } else {  
 return done(null, false);  
 }  
 });  
 }  
 }))  
 } catch (e) {  
 throw (e);  
 }  
 }  
}))

* + 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.

**// staff.js**  
***router***.get('/', function (req, res, next) {  
 if (isStaff(req, res)) {  
 res.render('dashboard\_staff');  
 } else {  
 res.sendFile(path.join(\_\_dirname, '../public', 'view\_customers.html'));  
 }  
});

function isStaff(req, res) {  
 if (isAuthenticated(req)) {  
 const user = getUser(req);  
 return user.role !== 3  
 }  
 res.redirect('/users/login');  
}

**// auth.js**  
function getUser(req) {  
 if (req.isAuthenticated()) {  
 let property = 'user';  
 if (req.\_passport && req.\_passport.instance) {  
 property = req.\_passport.instance.\_userProperty || 'user';  
 }  
 return req[property][0];  
 }  
}  
  
function isAuthenticated(req) {  
 return req.isAuthenticated()  
}

1. **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.

|  |  |  |
| --- | --- | --- |
| **A picture containing graphical user interface, application  Description automatically generated** | **A picture containing graphical user interface, application  Description automatically generated** | **A picture containing table  Description automatically generated** |

* **Login error messages:**

|  |  |
| --- | --- |
| **Graphical user interface, application  Description automatically generated** | **Graphical user interface, application  Description automatically generated** |

* + **Addition of new medicine(s) and supplier(s) are rejected** if their corresponding data types are invalid.

1. **Future Improvements**
   * 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
2. **Code Base**

* **Github Link:** [**https://github.com/piumallick/DeliveRx**](https://github.com/piumallick/DeliveRx)
* **Database has been hosted in Cloud through Aiven:** [**https://aiven.io/postgresql**](https://aiven.io/postgresql)

1. **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.

1. **References**

|  |  |  |
| --- | --- | --- |
| SL. No. | Functionality | Reference URL |
|  | Home Page (CSS Template) | [**https://github.com/john-smilga/js-cart-setup/tree/master/js**](https://github.com/john-smilga/js-cart-setup/tree/master/js) |
|  | Login & Registration Functionality | * [**https://medium.com/@timtamimi/getting-started-with-authentication-in-node-js-with-passport-and-postgresql-2219664b568c**](https://medium.com/@timtamimi/getting-started-with-authentication-in-node-js-with-passport-and-postgresql-2219664b568c) * [**https://reallifeprogramming.com/node-authentication-with-passport-postgres-ef93e2d520e7**](https://reallifeprogramming.com/node-authentication-with-passport-postgres-ef93e2d520e7) * [**https://www.youtube.com/watch?v=vxu1RrR0vbw**](https://www.youtube.com/watch?v=vxu1RrR0vbw) |
|  | Font Styling | * [**https://fontawesome.com/**](https://fontawesome.com/) |