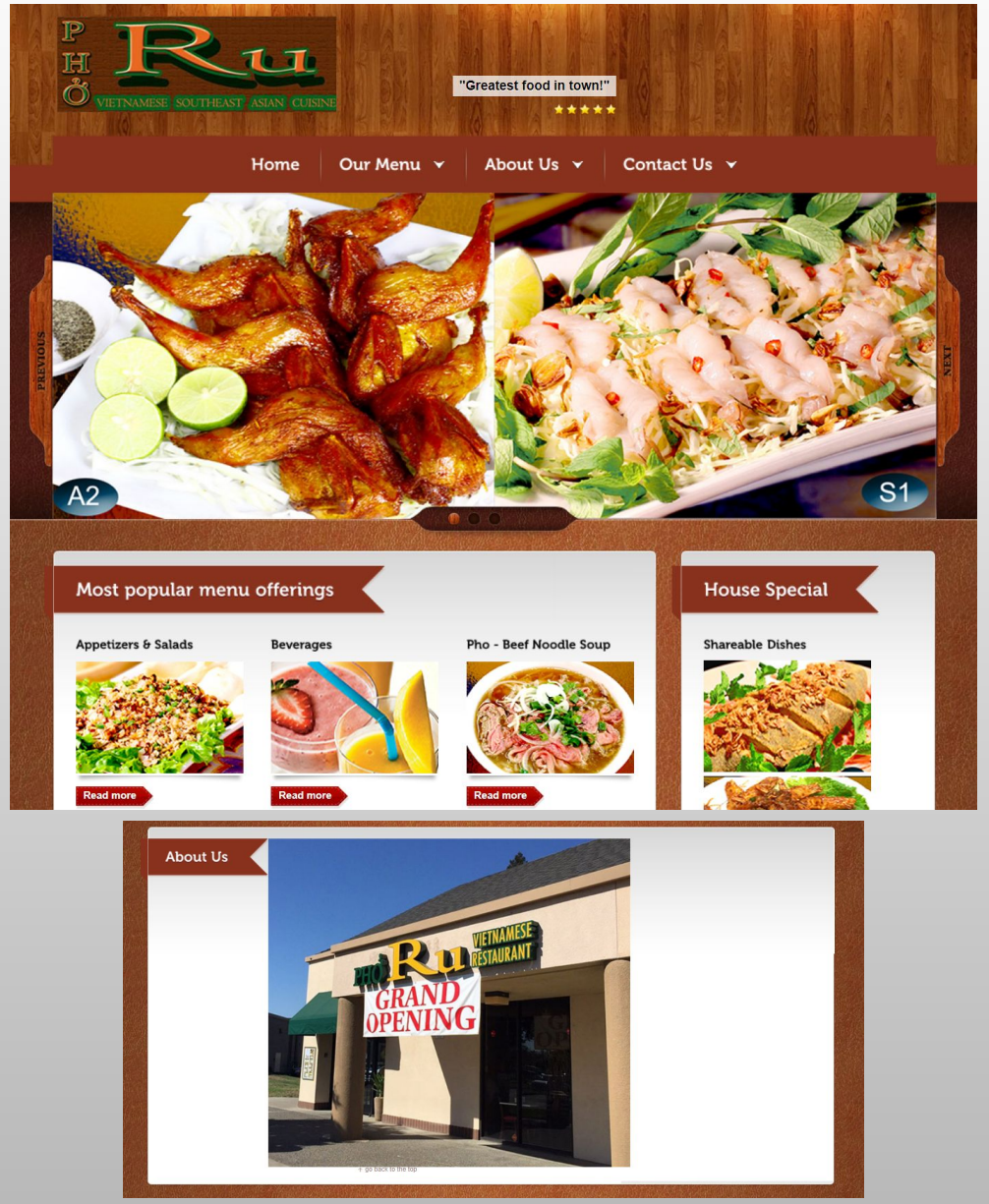
Pho Ru Website

Team Energy: Misba Tabassum, Massoud Bakhtyari,
Naba Riaz, Zoha Alvi, Nam Tran, Duc Nguyen

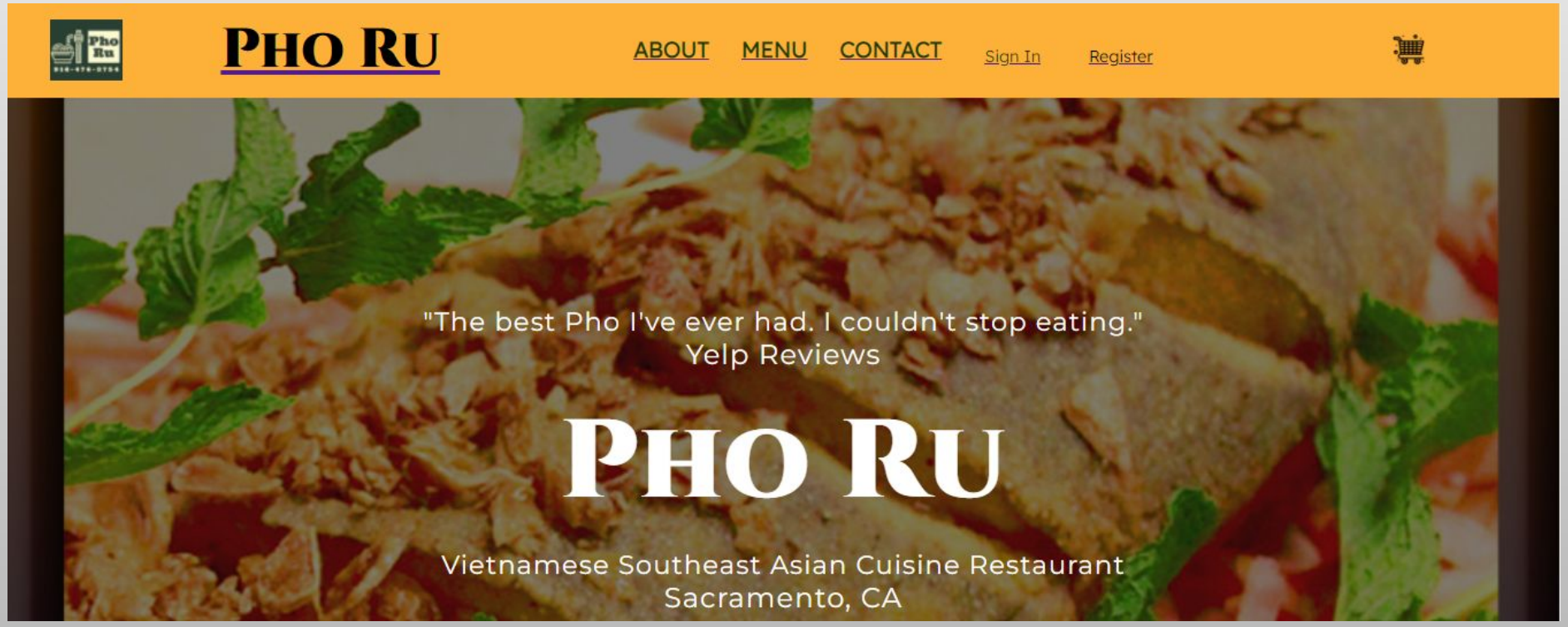
1. Project Overview

- Client
  - Lynn Nguyen, owner of Pho Ru
- Client's Business:
  - A Vietnamese Restaurant in Sacramento
- Client's "problem":
  - Outdated website
    - Incorrect information (menu items, restaurant hours, pricing, etc.)
    - User Interface (UI) doesn't reflect the owner's current vision
    - Customers are unable to order online
- Proposed Solution
  - Customers can create an account
  - Customers can order online
  - Update/fix information
  - Better UI

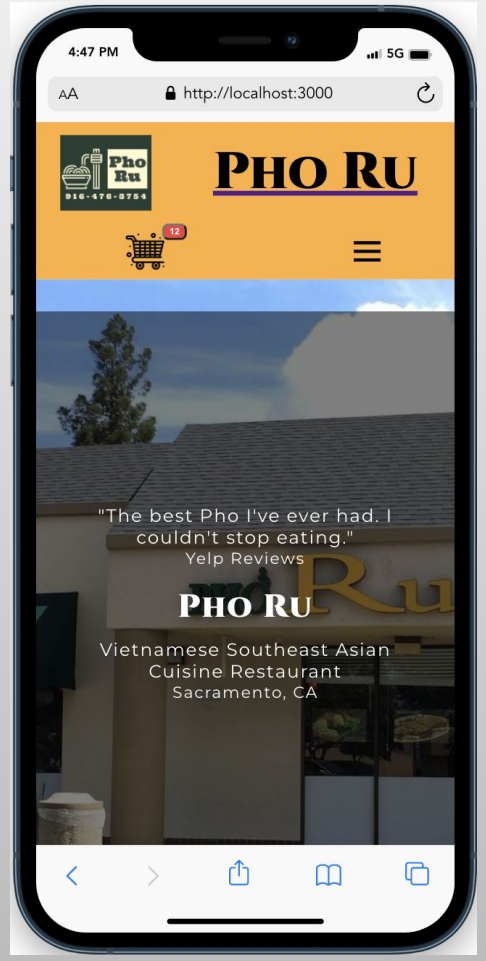


2. Product Web Demo

- Link to the deployed website: https://626883441640c715162ae8f9--thriving-creponne-382db8.netlify.app/



2. Product Mobile View Demo



3. Design - Register

- Bcrypt: password hashing
- Sequelize

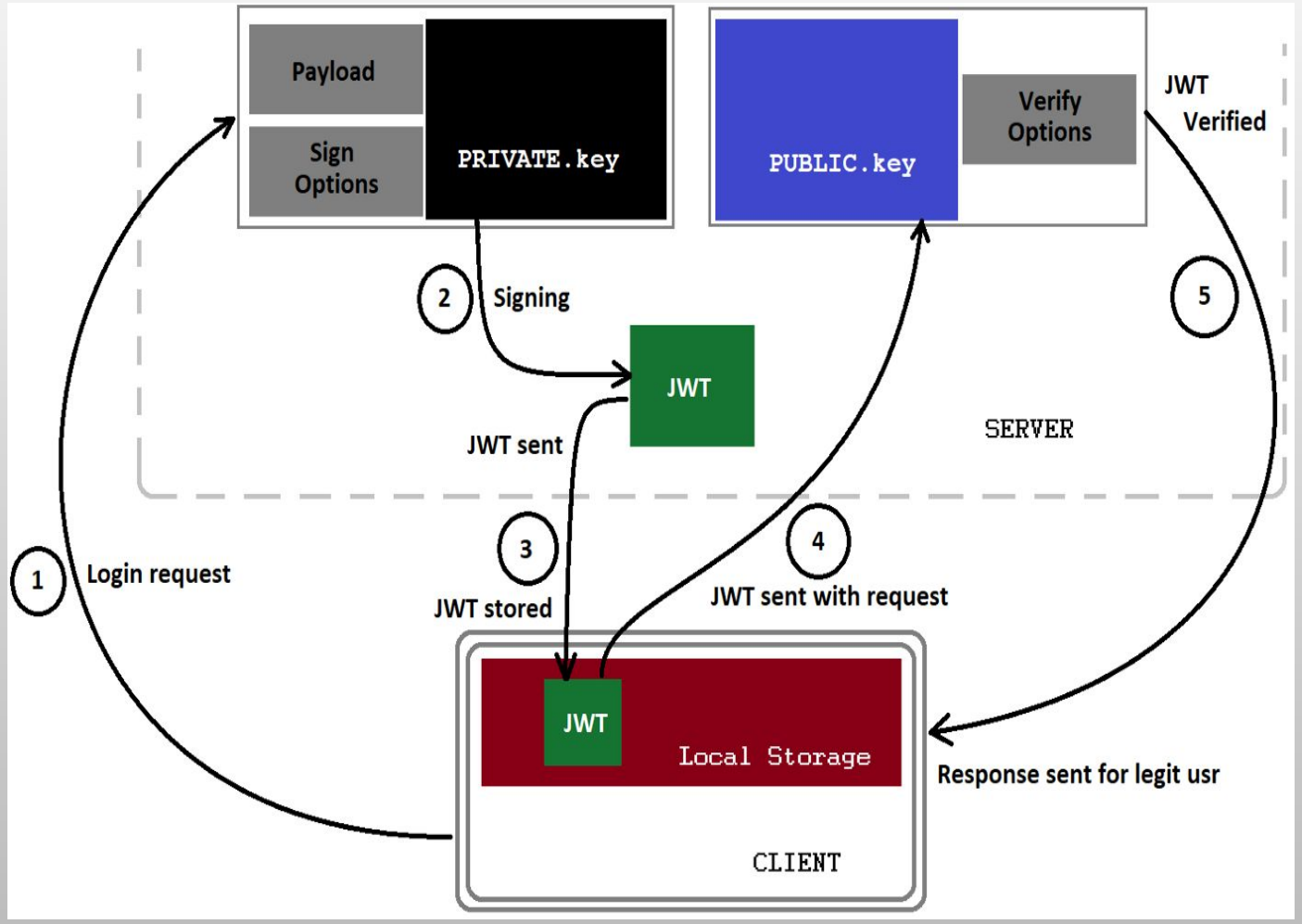
```
//- For register
router.post("/", async (req, res) => {
  const { name, last_name, username, password, question_one, questio
  const user = await Users.findOne({ where: { username: username } }
  if(user) {
     res.json({ error: "User already Exist." })} else {
  bcrypt.hash(password, 10).then((hash) => {
    Users.create({
      name: name,
      last_name: last_name,
      username: username,
      password: hash,
      question_one: question_one,
      question_two: question_two,
    });

    res.json("Success");
  });
  }
});
```

```
module.exports = (sequelize, DataTypes) => {
        const Users = sequelize.define("Users", {
                name: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
                last_name: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
                username: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
                password: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
                question_one: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
                question_two: {
                        type: DataTypes.STRING,
                        allowNull: false,
                },
```

id	name	last_name	username	password	question_one	question_two	createdAt	updatedAt
2	Massoud	Bakhtyari	massoud.bakhtyari@yahoo.com	$2b$10$1zQbOkVOOSWexhyBKCNKZ.F9qFJy....	dsads	CARMICHAEL	2022-04-21 05:41:17	2022-04-21 05:41:17
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Design - Sign In

Current Presenter:
Massoud




```
const { verify } = require('jsonwebtoken');

//- Create a middleware (a function with req, res, and next)
//-    . req: get stuff from the request
//-    . res: send stuff back
//-    . next: allow the request to move forward
const validateToken = (req, res, next) => {
  //-- get the access token being passed in the header
  const accessToken = req.header('accessToken');

  //-- if user are not logged in (no access token found),
  //--    send back an error object
  if (!accessToken) {
    console.log(accessToken);
    return res.json({ error: 'User not logged in!' });
  }
```

```
router.post("/login", async (req, res) => {
  const { username, password } = req.body;

  //-- Check if the user existed in our database
  //--   . if: user not exist, send error feedback
  //--   . else: user exists, check password
  //--     . if: password not match, send error feedback
  //--     . else: password correct, create and return token
  const user = await Users.findOne({ where: { username: username } });

  if (!user) {
    res.json({ error: "User Doesn't Exist." });
  } else {
    bcrypt.compare(password, user.password).then((match) => {
      if (!match) {
        res.json({ error: "Invalid Password." });
      } else {
        //-- Create token: pass in the data that needs to be secured.
        const accessToken = sign(
          { username: user.username, id: user.id },
          "[redacted]"
        );
        res.json(accessToken);
      }
    });
  }
});
```

3. Design - Profile Page,Sign out

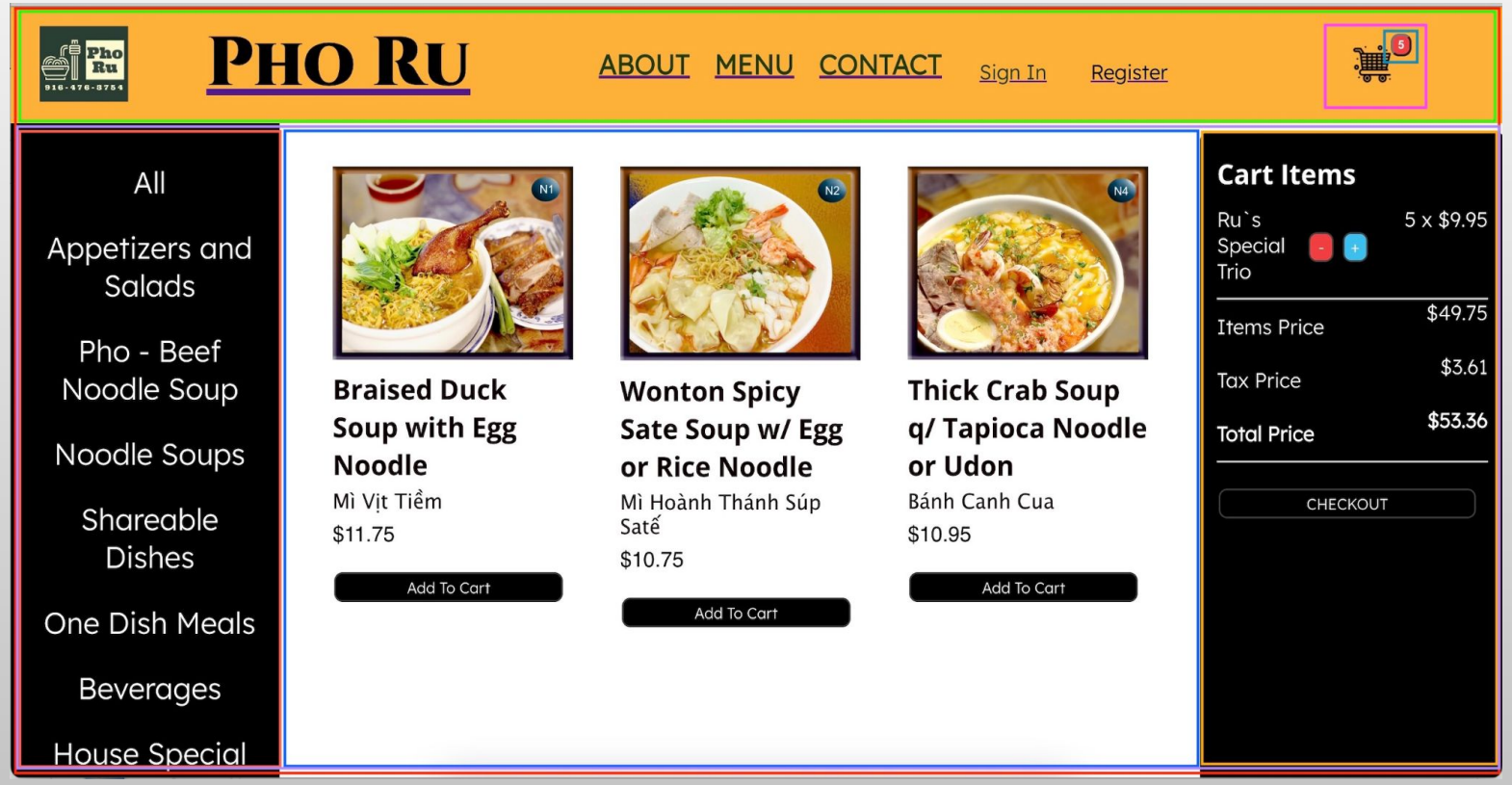
Current Presenter:
Massoud


```
const logout = () => {
  localStorage.removeItem("accessToken");
  navigate("/");
  setAuthState(false);
};
```

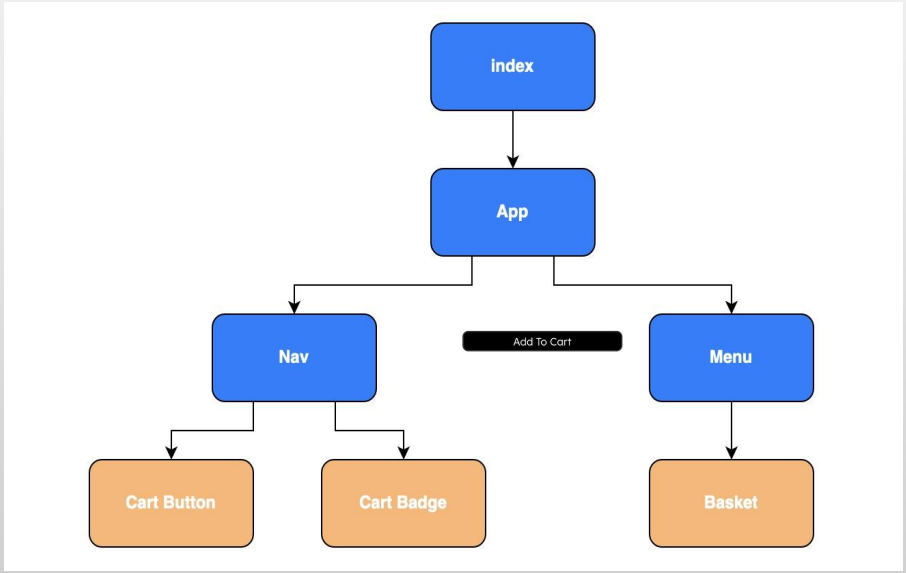
3. Design - Shopping Cart

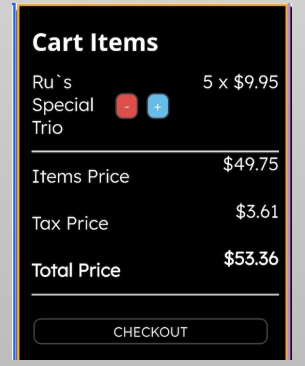
Current Presenter:
Duc

Multiple components are used on one page.



Cart Button, Add-to-cart Button and Cart Items are on different components





How to keep track of user interaction with those elements and update the cart items correctly?

3. Design - Shopping Cart cont.

Current Presenter:
Duc

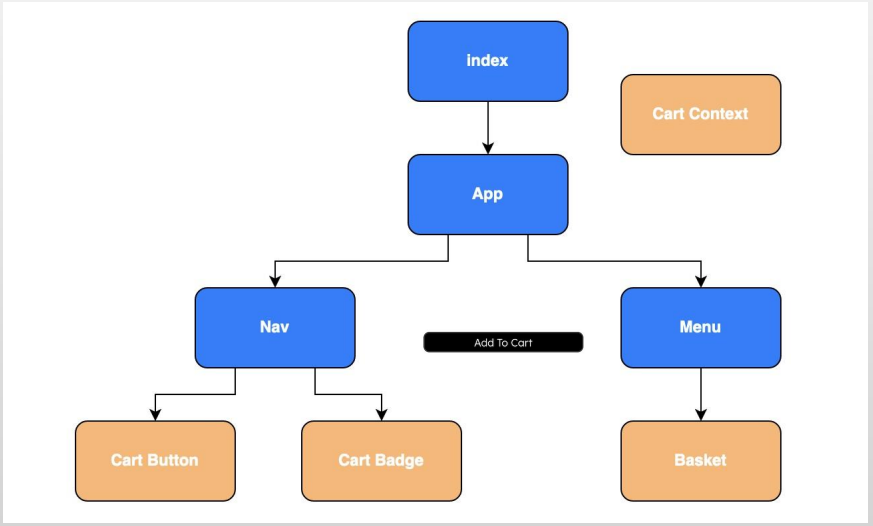
Cart Context: store global states and function

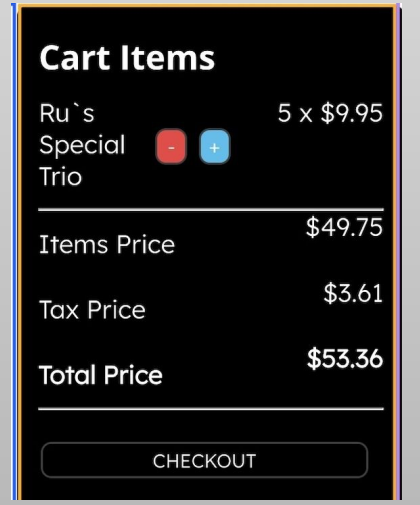
```
index.js  CartContext.js  CartContextSample.js 5  App.js  PaymentSc
Temp > CartContextSample.js > CartContextProvider
1  import { createContext, useEffect, useState } from 'react';
2
3  //-- Create a Context for Cart
4  export const CartContext = createContext();
5
6  export const CartContextProvider = (props) => {
7    //-- Tracking states For Basket: tracking Cart's items
8    const [cartItems, setCartItems] = useState(...);
9    const [totalCartItems, setTotalCartItems] = useState(...);
10   const [isCartClicked, setIsCartClicked] = useState(false);
11
12   //-- Functions for Cart
13   const onAddToCart = (item) => {...};
14
15   const onRemoveFromCart = (item) => {...};
16
17   const toggleCartClicked = () => {...}
18
19   // const [state, dispatch] = useReducer(reducer, initialState);
20   const value = {
21     cartItems: cartItems,
22     totalCartItems: totalCartItems,
23     isCartClicked: isCartClicked,
24     onAddToCart: onAddToCart,
25     onRemoveFromCart: onRemoveFromCart,
26     toggleCartClicked: toggleCartClicked,
27     setCartItems: setCartItems,
28     setTotalCartItems: setTotalCartItems
29   }
30
31   return <CartContext.Provider value={value}>{props.children}</CartContext.Provider>
32 }
33
```

Share the global states and functions with the App component

```
index.js
frontend > restaurant > src > index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import { CartContextProvider } from './helpers/CartContext';
4  import { AuthContextProvider } from './helpers/AuthContext';
5  import './index.css';
6  import App from './App';
7  import reportWebVitals from './reportWebVitals';
8
9  ReactDOM.render(
10   <React.StrictMode>
11     <AuthContextProvider>
12       <CartContextProvider>
13         <App />
14       </CartContextProvider>
15     </AuthContextProvider>
16   </React.StrictMode>,
17   document.getElementById('root')
18 );
```

```
CartContextSample.js 5  MenuSample.js  Menu.js  App.js  PaymentScreen.js
Temp > MenuSample.js > default
1  import { useState, useContext } from 'react';
2  import Basket from '../components/Basket';
3  import { CartContext } from '../helpers/CartContext';
4
5  function Menu() {
6
7    //-- Get Context value from CartContext in index.js
8    const { cartItems, onAddToCart, onRemoveFromCart, isCartClicked } = useContext(CartContext);
9
10   return (
11     <div>
12       {
13         isCartClicked ? (
14           <Basket
15             cartItems={cartItems}
16             onAddToCart={onAddToCart}
17             onRemoveFromCart={onRemoveFromCart}
18           />
19         ) : (
20           ''
21         )
22       }
23     </div>
24   );
25 }
26
27 export default Menu;
```

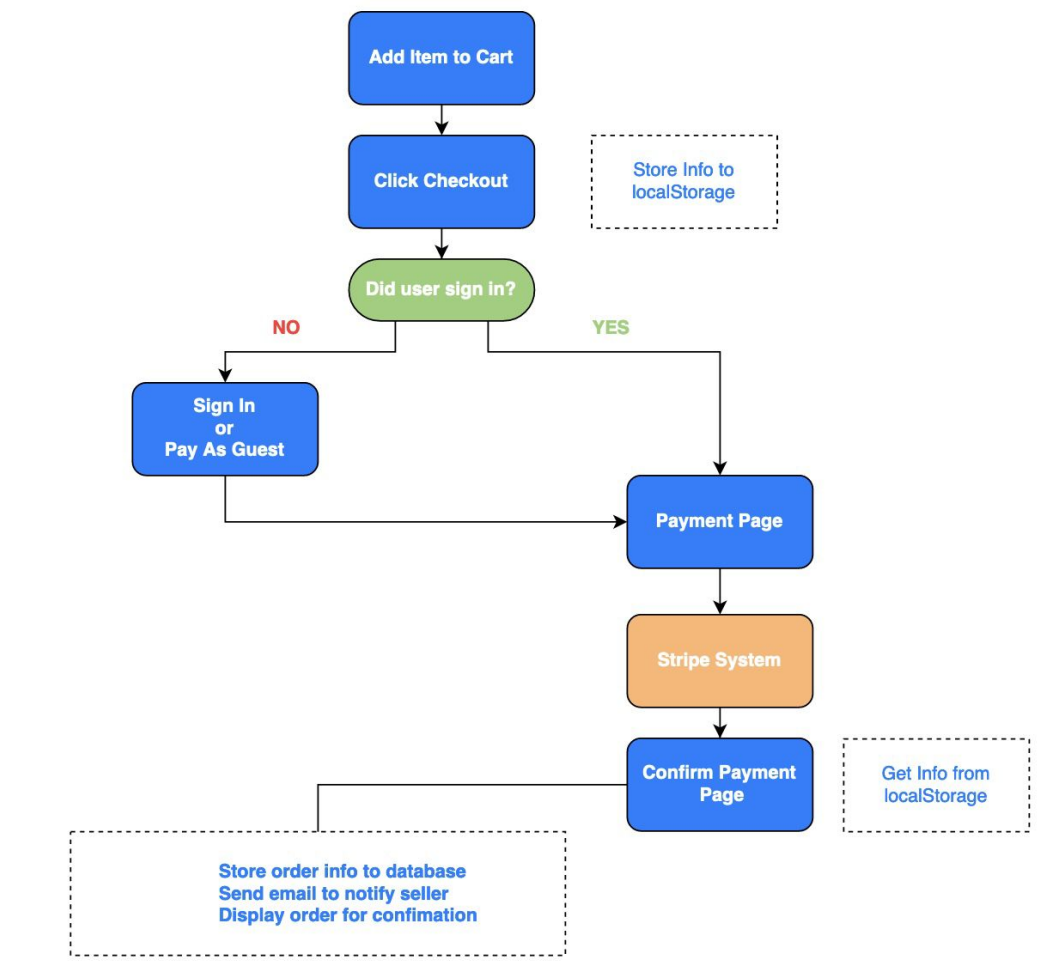




Use isCartClicked to toggle the Basket

3. Design - Checkout Process

Current Presenter:
Duc



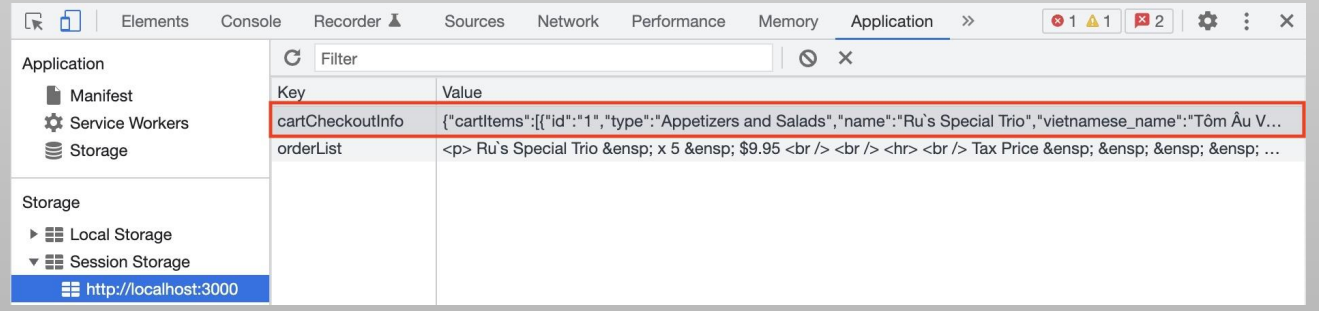
Send payment info to backend

```
if (!error) {
  try {
    const { id } = paymentMethod
    const response = await axios.post("http://localhost:3001/payment", {
      amount: d,
      description: cartItems,
      id
    })

    if (response.data.success) {
      console.log("Successful payment")
      setPaymentInProgress(false)
      setSuccess(true)

    }

  } catch (error) {
    console.log("Error", error)
  }
} else {
  console.log(error.message)
}
}

useEffect(() => {
  if (success) {
    navigate('/confirmpayment')
  }
}, [success])
```

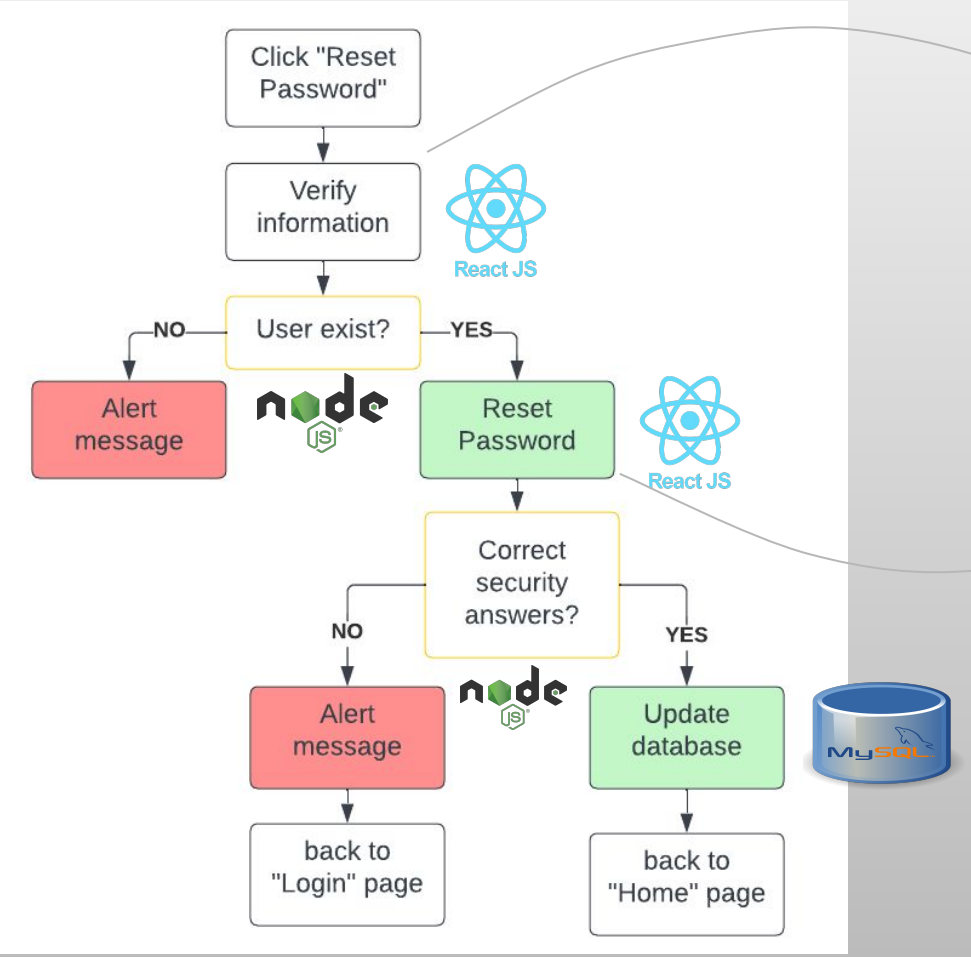
EmailJS

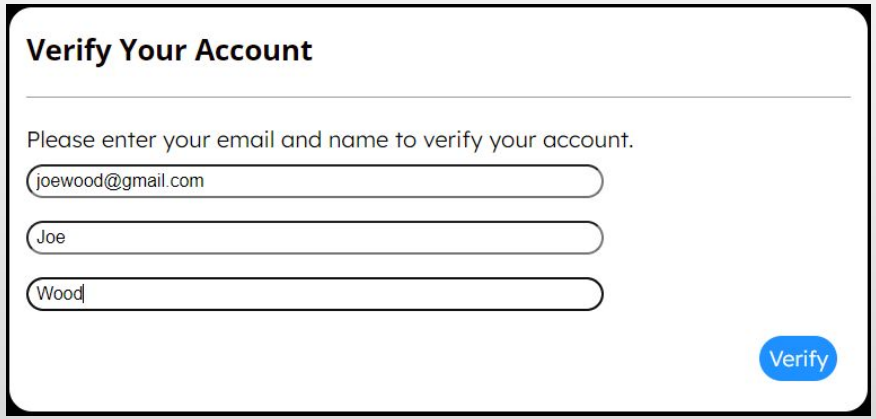
stripe

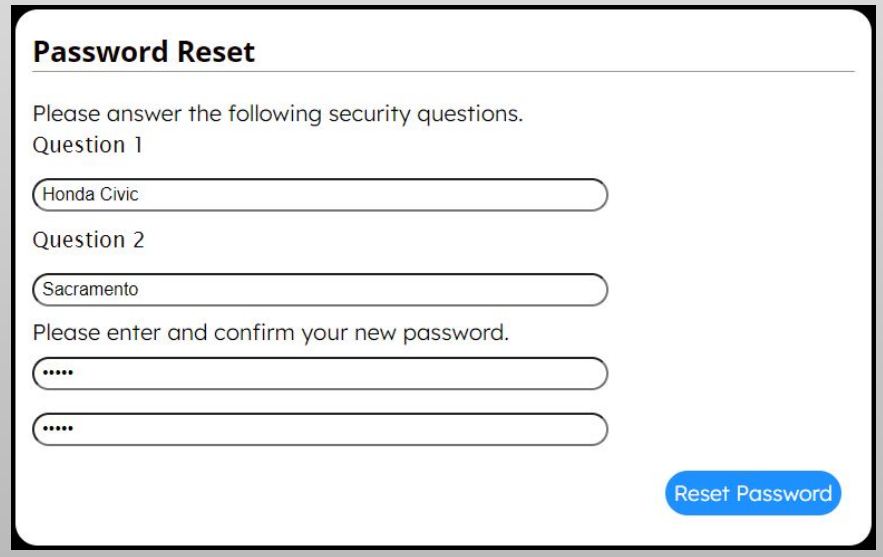
Store string to local storage



3. Design - Reset Password





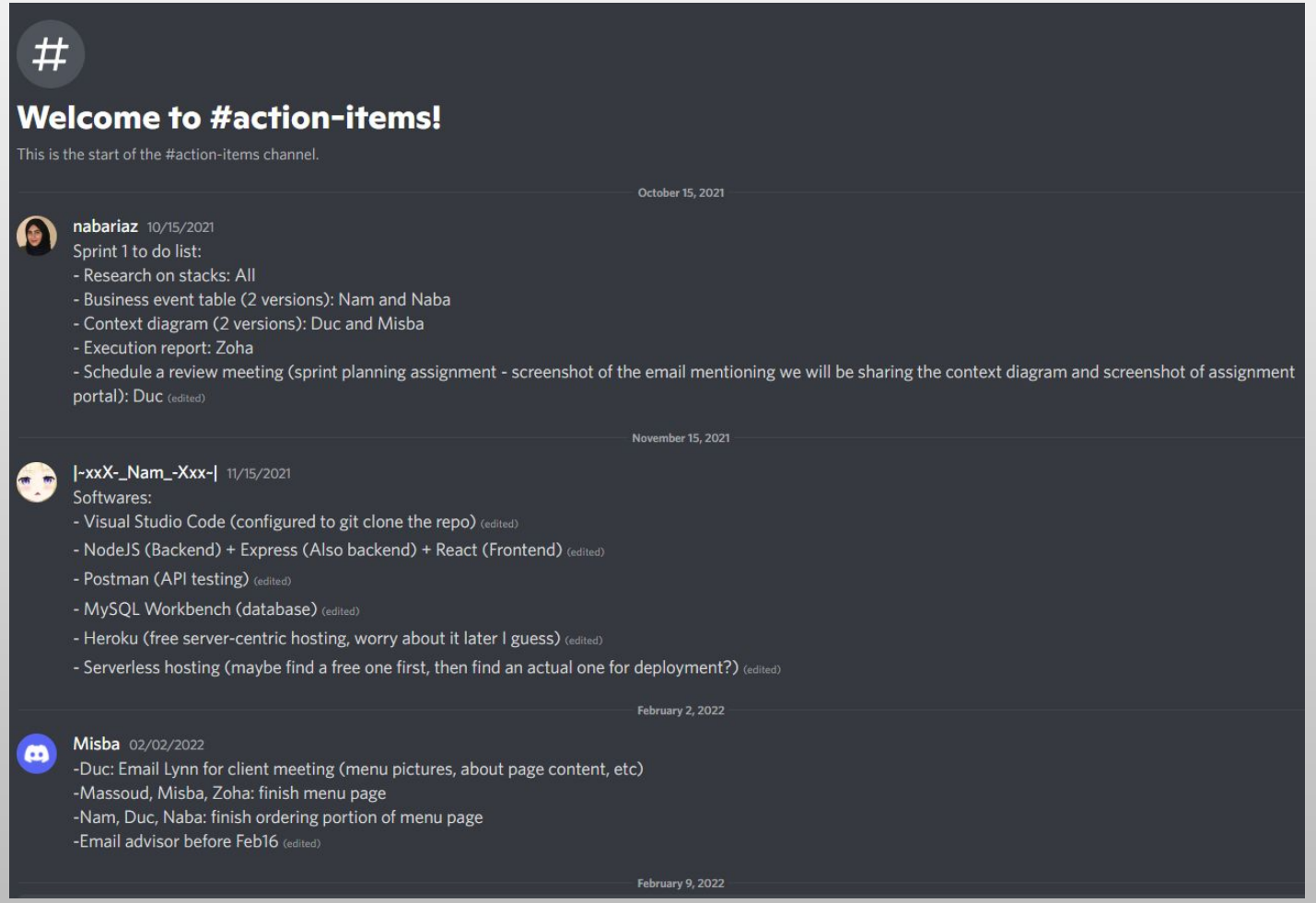


4. Technologies and SCRUM

Current Presenter:
Misba

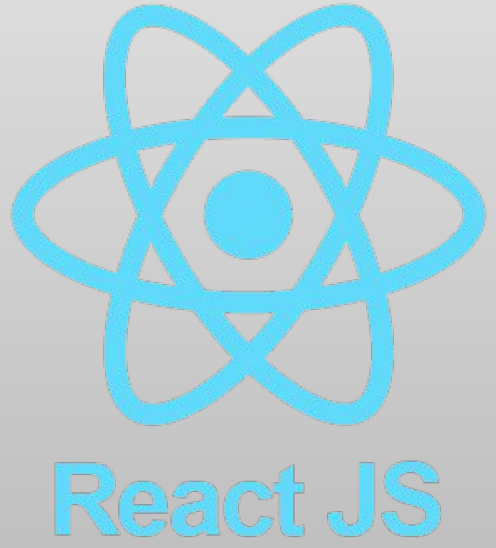
- SCRUM Roles
  - Product Owner, Scrum Master, and Scrum Team worked in collaboration with stakeholder

- SCRUM Practices Used
  - Sprint planning and refining requirements
  - Managing backlog and prioritizing work
  - Weekly standup meetings and daily communication via Discord or Zoom
  - Retrospectives



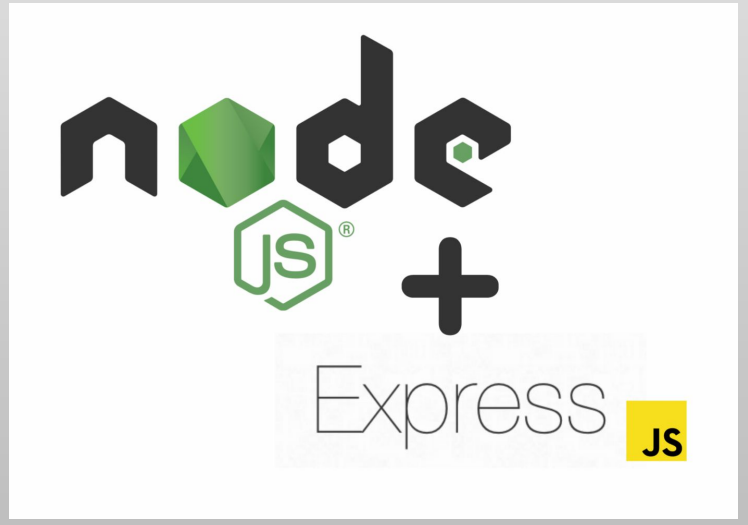
4. Technologies – Frontend

- React
- HTML/CSS



4. Technologies – Middleware

- Express
- Axios



4. Technologies – Backend

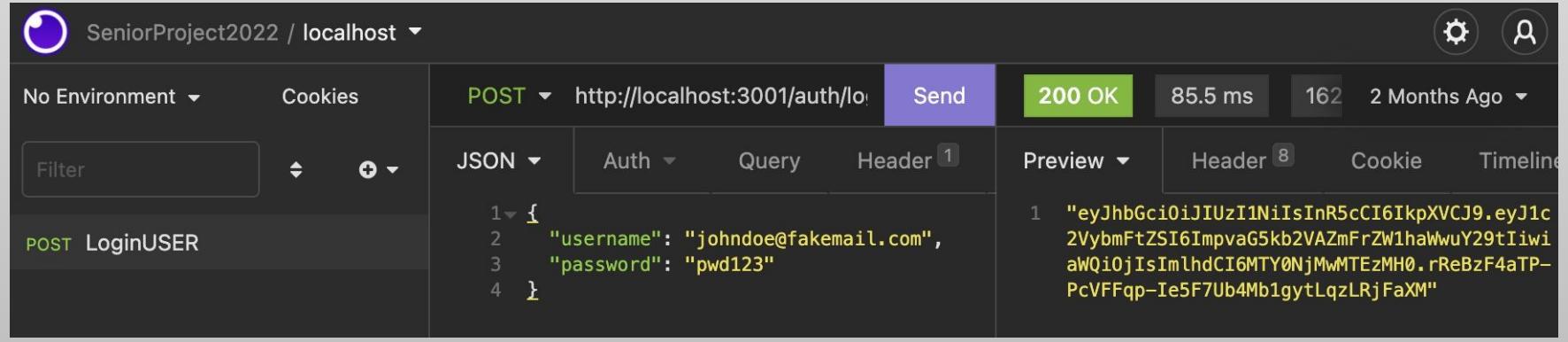
- Sequelize
- MySQL

4. Technologies – Other Tools

VS Code, MySQL Workbench, GitHub, PostMan, Insomnia, Selenium

5. Testing

- Functional testing: Manually
- API requests: Insomnia, Postman
- Integration Testing after each sprint
  - Peer Review
  - Make use of GitHub



5. Testing – Selenium Examples

- Selenium for automated test cases

```python
# Test File 01: Navigate between pages

#-- Import webdriver from selenium
from selenium import webdriver
import time

#-- Provide path to ChromeDriver
PATH = '/Users/ducnguyen/Programs/chromedriver'

#-- Select the driver we want to use, and specify the webdriver
#-- for this web browser
driver = webdriver.Chrome(PATH)
driver.maximize_window()

#-- Open PhoRu website
driver.get('http://localhost:3000')

#-- To get the title of the website
print(driver.title)

#-- Check About Page button
time.sleep(2)
about = driver.find_element_by_class_name('about-btn')
about.click()
time.sleep(2)
```

```python
#-- Check Cart button
time.sleep(2)
cart = driver.find_element_by_class_name('cart-btn-img')
cart.click()
time.sleep(2)
cart.click()

#-- Check Sign In button
time.sleep(2)
signin = driver.find_element_by_class_name('login-btn')
signin.click()

#-- Check Sign Up button
time.sleep(2)
signup = driver.find_element_by_class_name('login-btn-signup')
signup.click()


#-- To close the entire browser
time.sleep(3)
driver.quit()
```

6. Deployment

- Frontend: Netlify
- Backend: Heroku

COST:
- Netlify + Heroku: $100 + $7 = **$107**/month
- EmailJs: **$0** for 200 emails/month | **$4** for 1000 emails/month

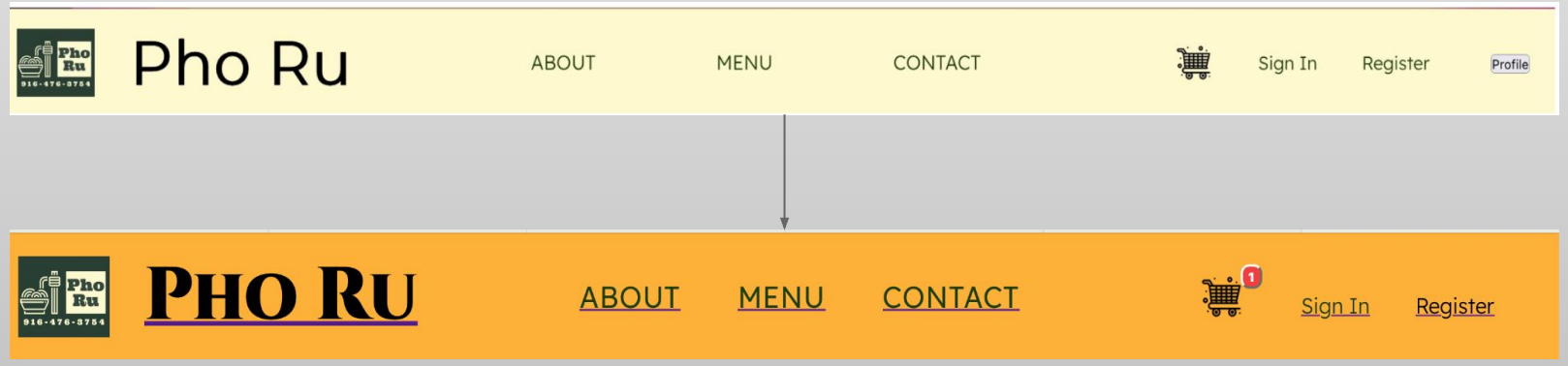
7. Lesson learned

- Things we learned
  - Full Stack Web-Development: React, Express, MySQL
  - New tools: Github, Postman, Jira, Selenium
  - Team Work, Communication, Self-learning
- Insights/wisdom
  - Perform regression testing after pushing code
  - Let team members know which part you are working on to avoid conflicting changes
  - Jira is a great organizational tool allowing us to organize stories and use the Kanban board
  - Set achievable goals and delegate tasks efficiently
  - Having the whole project planned out ahead of time is very helpful
  - Learn to adapt to sudden changes

8. Conclusion

- Client is very satisfied with the product so far
- Nothing to be improved thus far

Logo Changes



Navigation Bar Changes

Thank you for listening!