My GitHub Repository URL:

https://github.com/AgentMrCow/Restaurant-Ordering-Mobile-App/tree/main

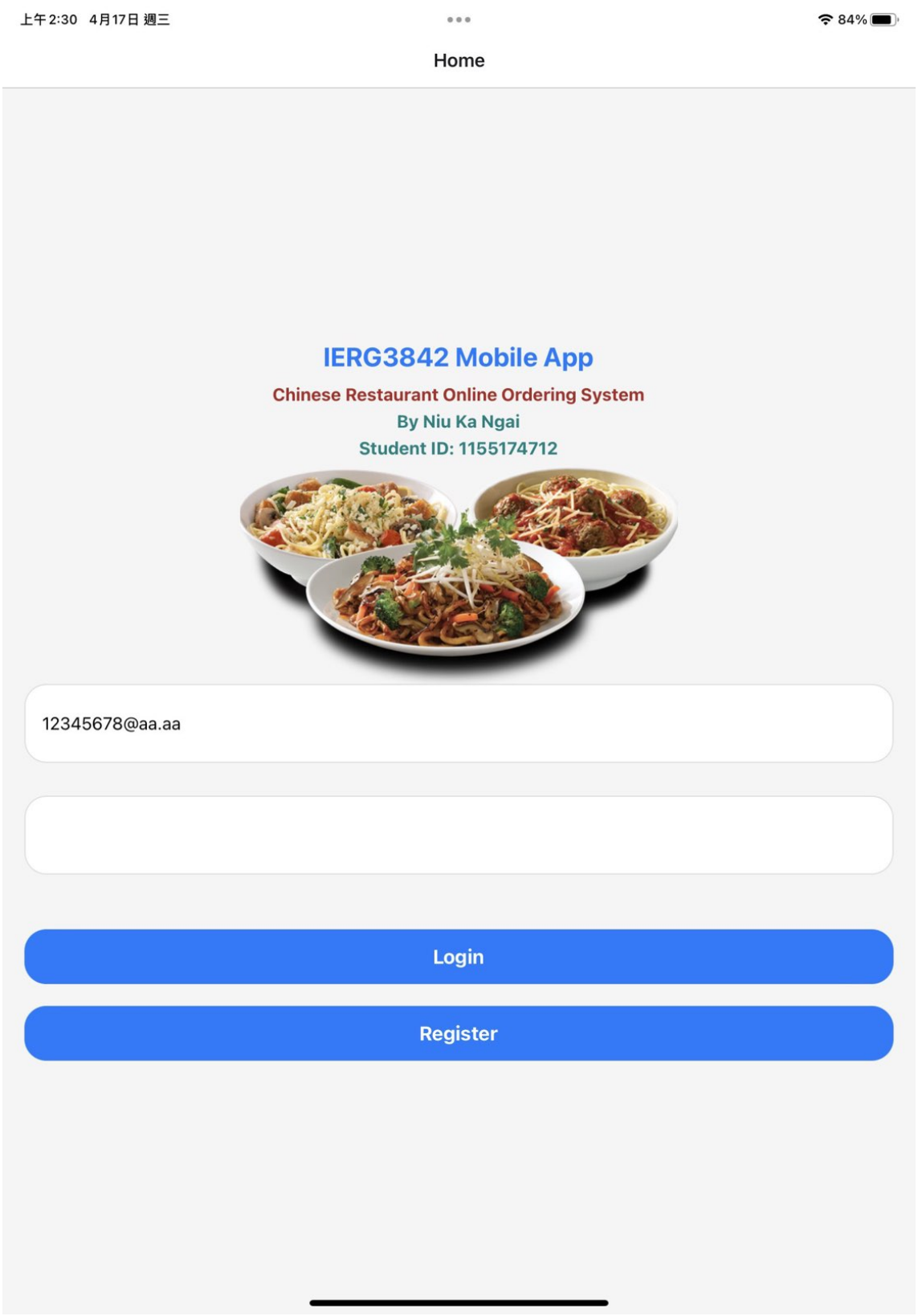
IERG3842 Mobile App - Chinese Restaurant Online Ordering System

Welcome to the IERG3842 Mobile App, a streamlined online ordering system designed specifically for a delightful Chinese cuisine experience. Developed by Niu Ka Ngai 1155174712, this app allows customers to easily register, log in, and place orders for their favorite Chinese dishes, all from the comfort of their smartphone.

App Screenshots

Here are some screenshots that illustrate the features and user interface of the IERG3842 Mobile App:

ipad Screen



This app can work perfectly in both Andorid and IOS environment. For convenience, I will only demostrate other screens in Android smartphone Pixel 3a.

Home Screen

4:19

Home

IERG3842 Mobile App

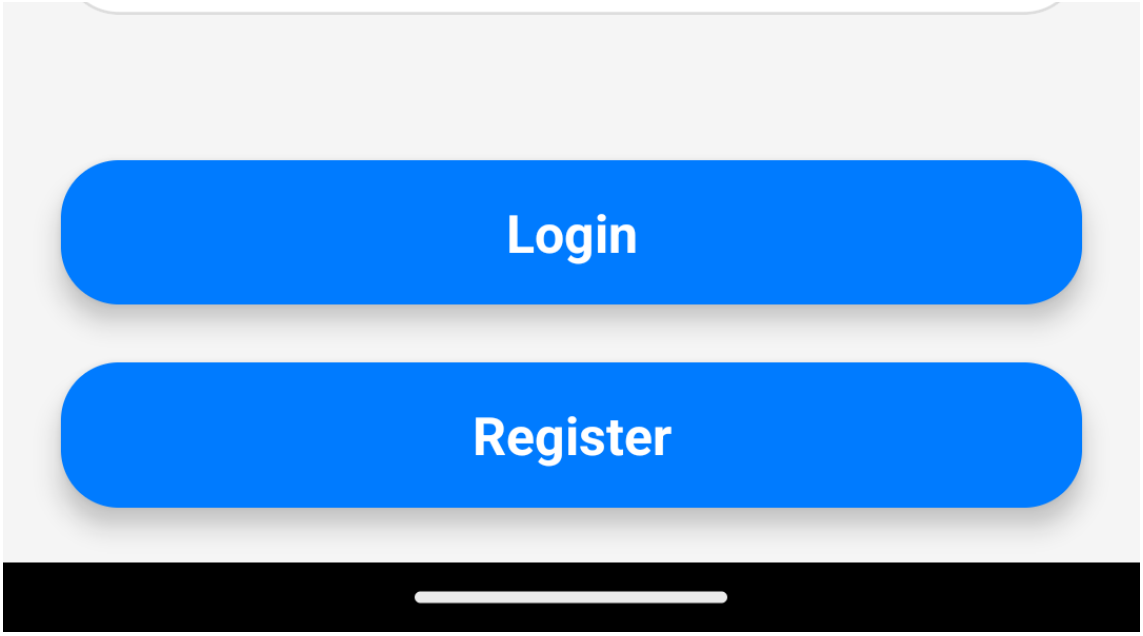
Chinese Restaurant Online Ordering System

By Niu Ka Ngai

Student ID: 1155174712

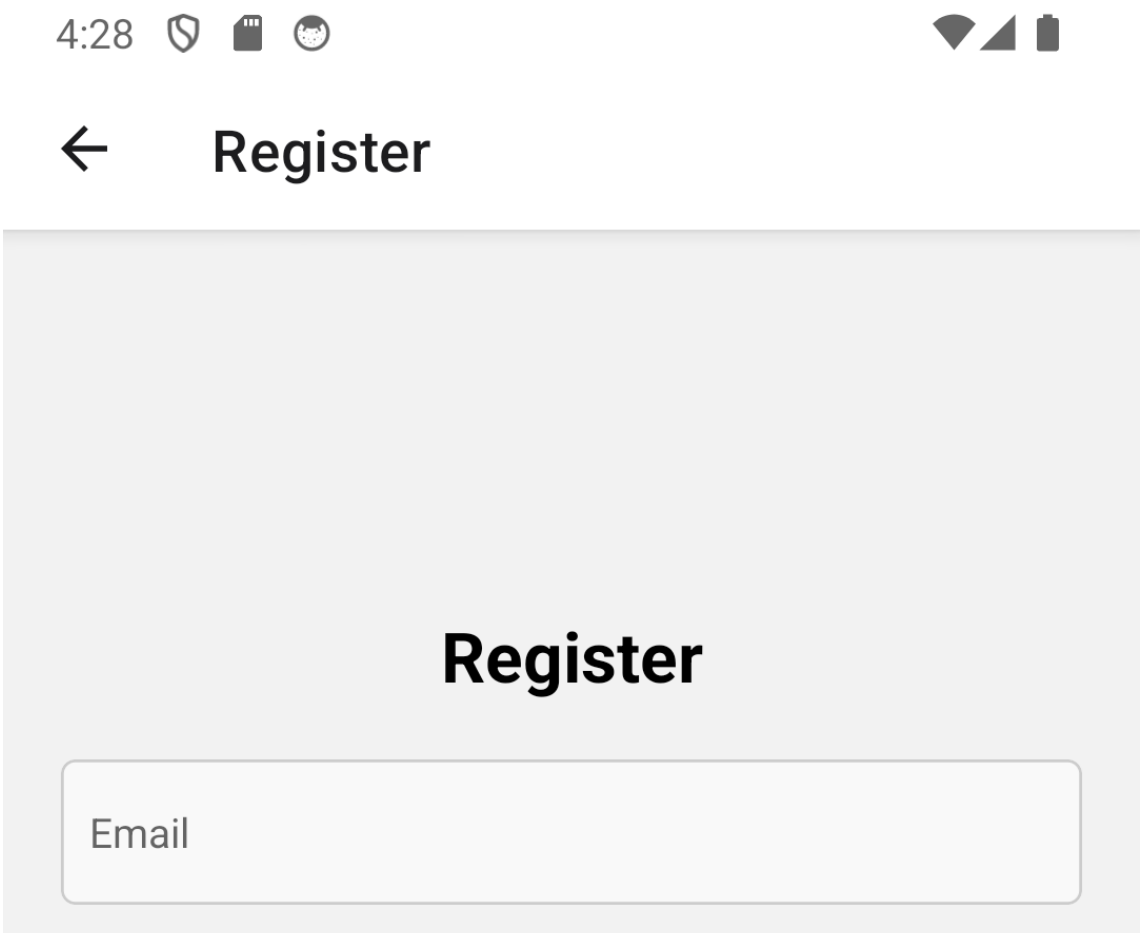
Email

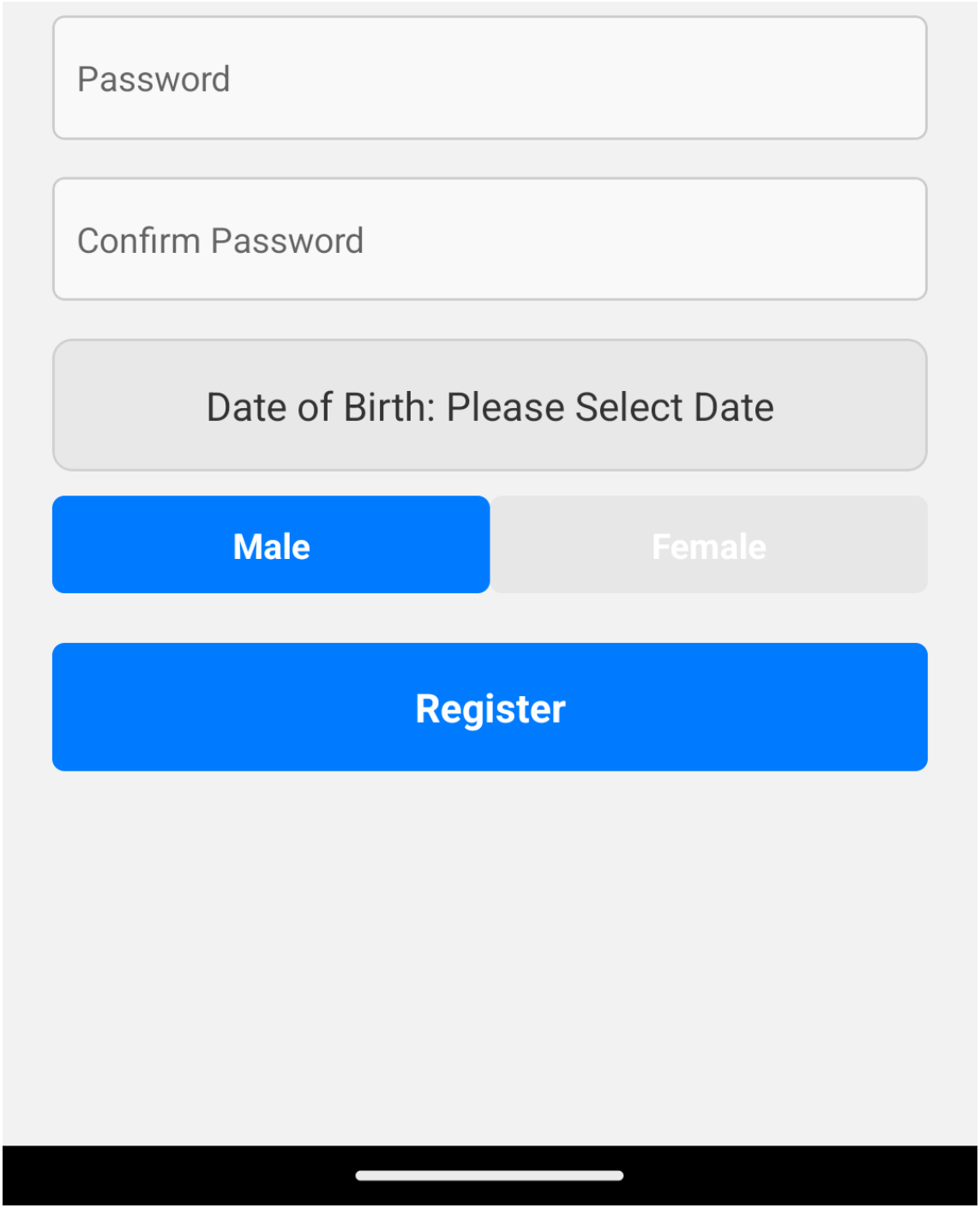
Password



This is the first screen users see when they open the app. It allows users to either log in to their existing account or register for a new account.

Registration Screen





New users can create an account by providing their email, password, and date of birth.

Cache

General Tso's Chicken - $12.99 x 2 (Subtotal: $25.98)

Remove

Peking Duck - $25.50 x 1 (Subtotal: $25.5)

Remove

Dim Sum Platter - $18.00 x 1 (Subtotal: $18)

Remove

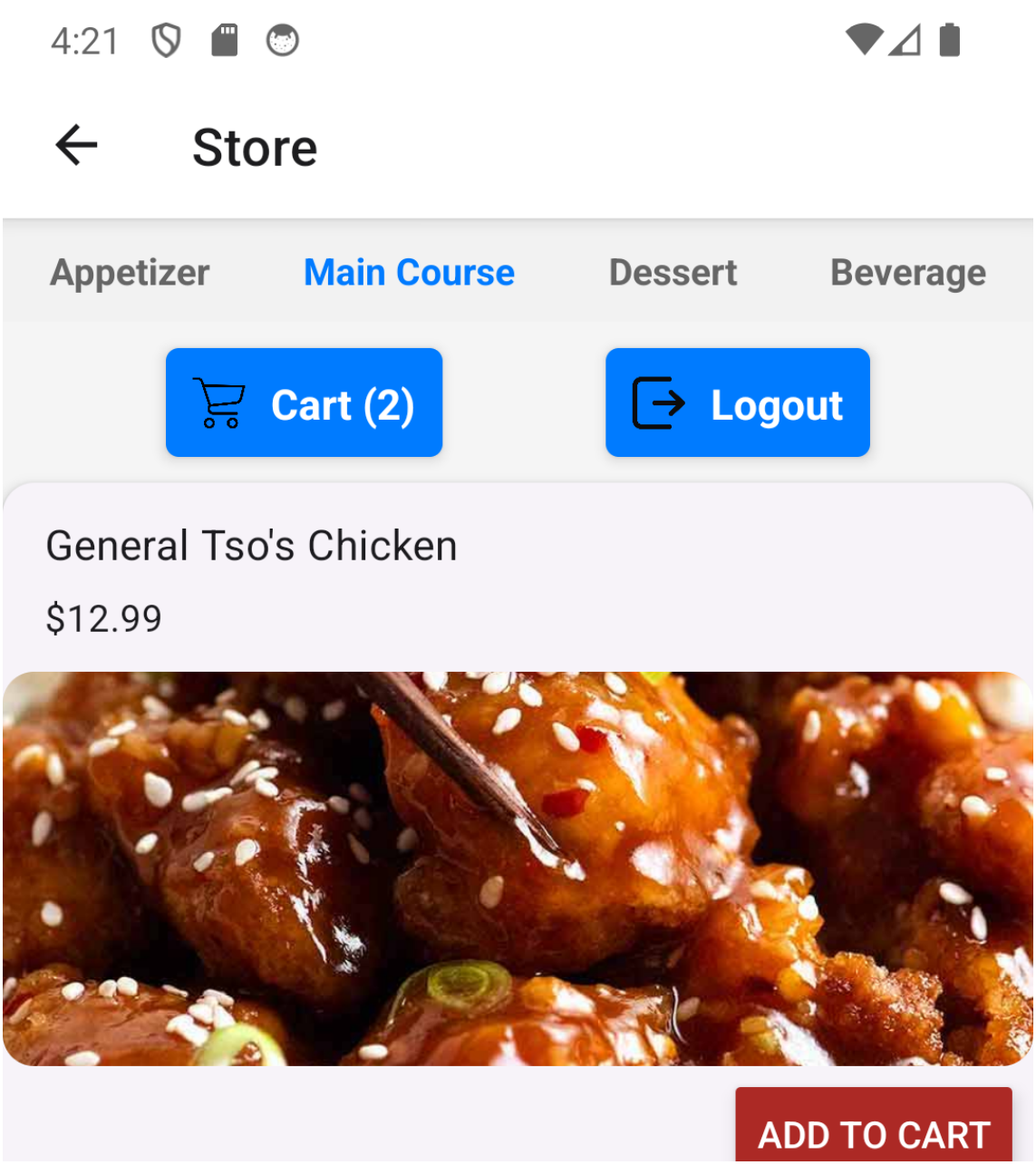
Total: $69.48

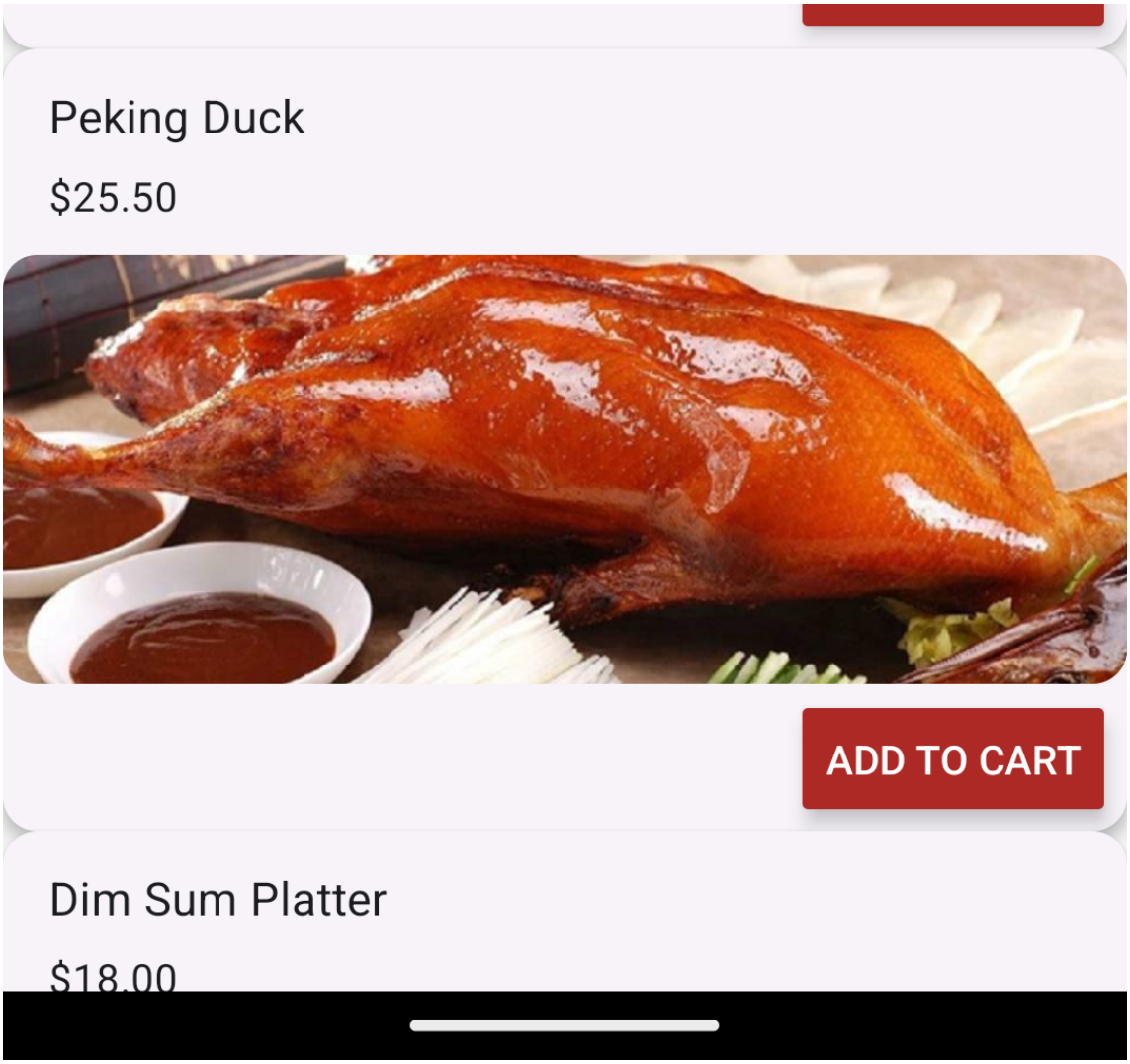
CONTINUE ORDERING

CHECKOUT

Once logged in, login tokens are saved in cache.

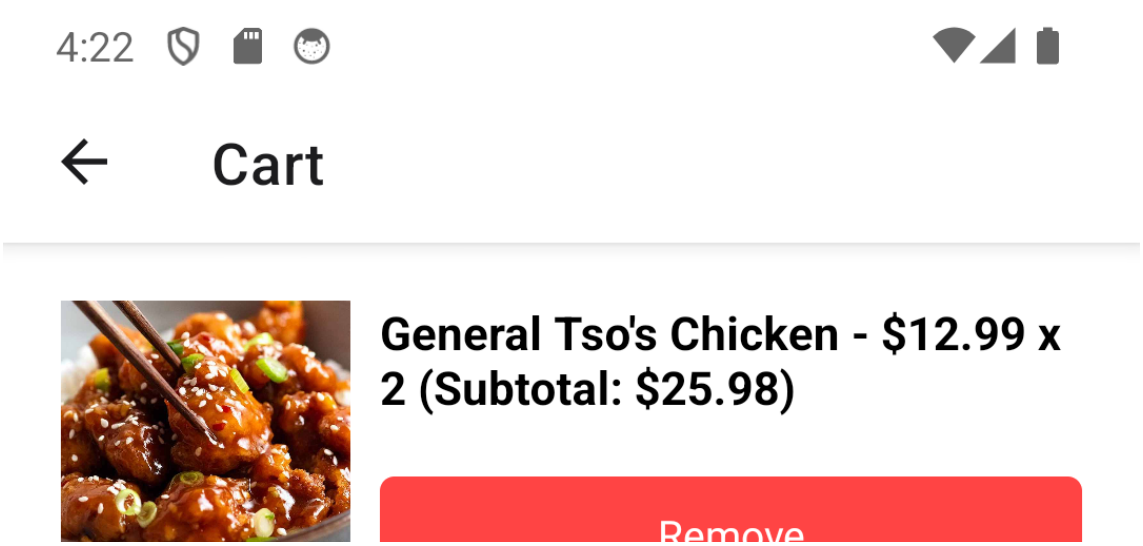
Store Page





Users can browse the menu and select items to add to their cart.

Shopping Cart



Remove

Peking Duck - $25.50 x 1
(Subtotal: $25.5)

Remove

Dim Sum Platter - $18.00 x 1
(Subtotal: $18)

Remove

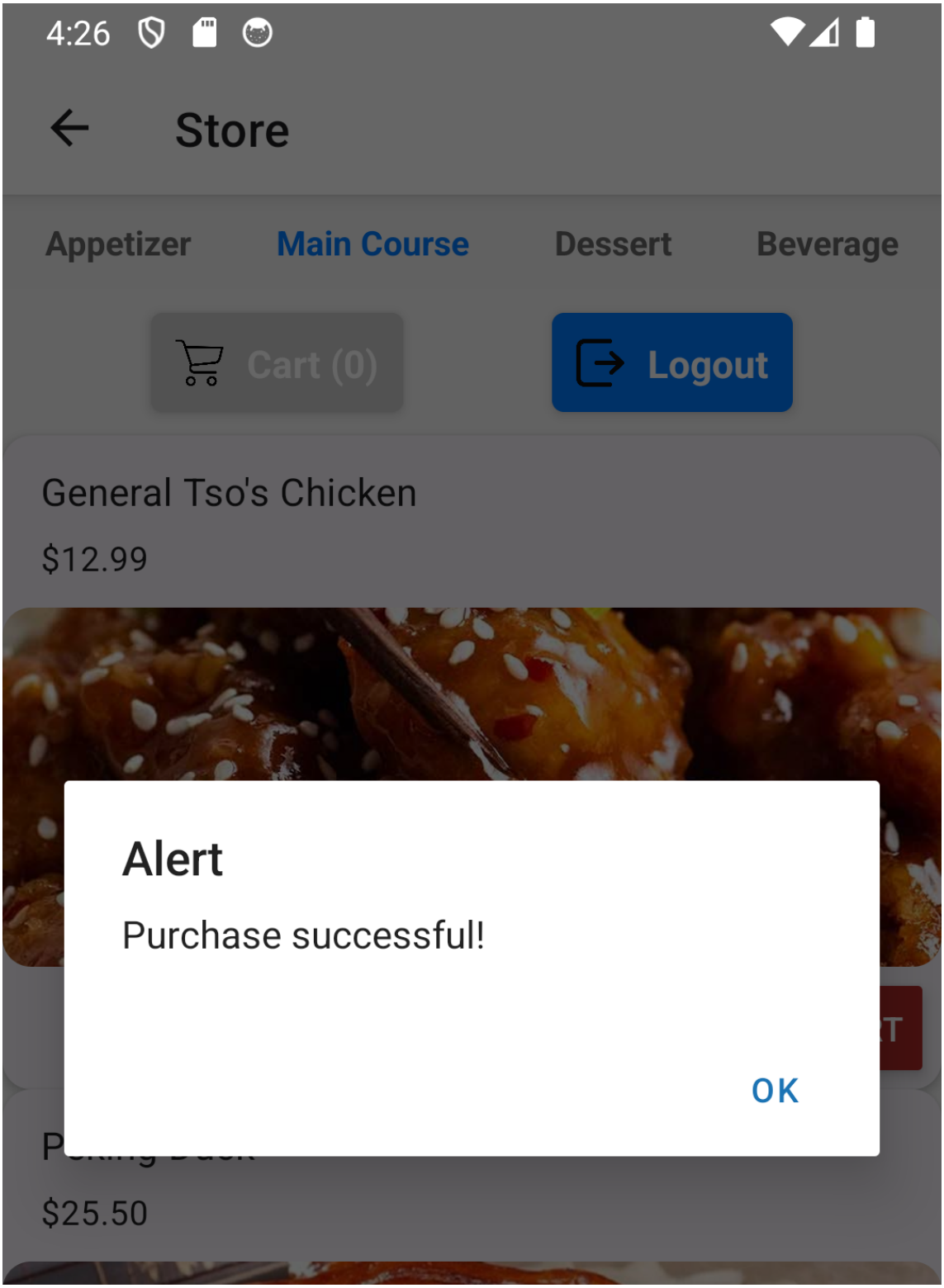
Total: $69.48

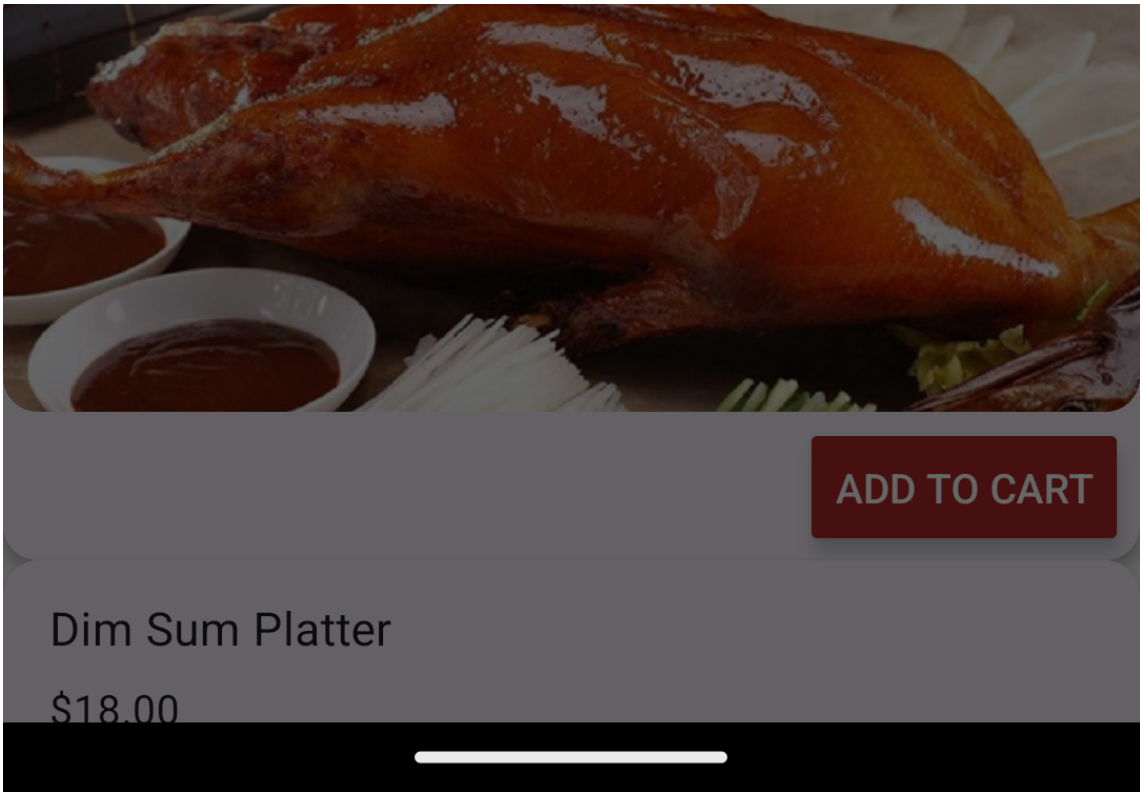
CONTINUE ORDERING

CHECKOUT

Users can remove or view the items they have added to their cart and proceed to checkout.

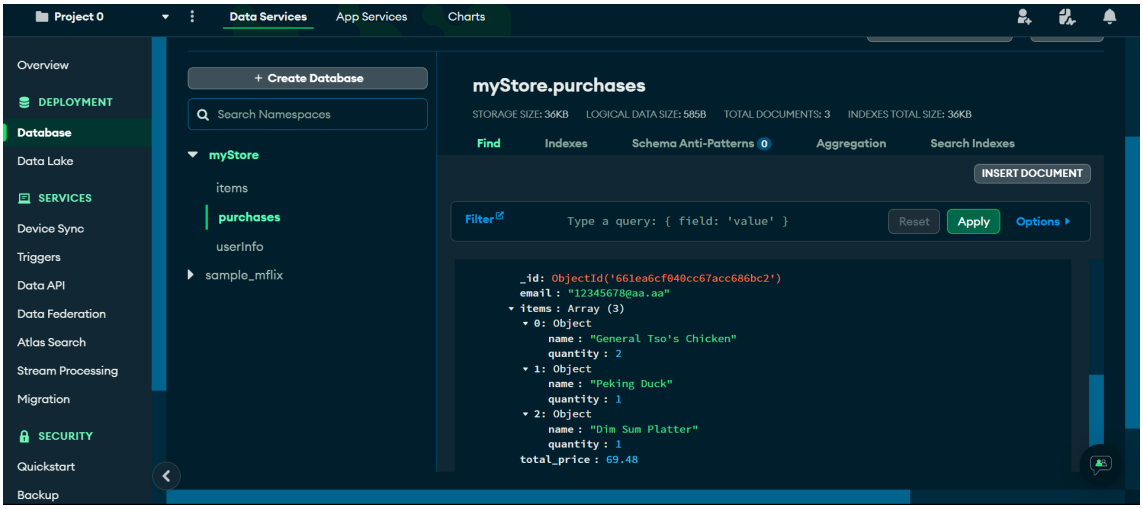
Successful Purchase





After a successful transaction, users receive a confirmation of their purchase.

Saving Order



The order is saved securely in MongoDB Atlas Database.

Core Technologies of the IERG3842 Mobile App

Here's an overview of the technologies and frameworks used to build this application, providing a robust and user-friendly experience:

1. Mobile Application Frontend

- **React Native**: Utilized for developing the cross-platform mobile app which allows it to run on both Android and iOS devices. React Native is a popular choice for mobile development due to its efficiency and the rich ecosystem of plugins and community support.
- **Expo**: A framework and platform for universal React applications. It is used to accelerate the development process by simplifying the setup and deployment of the React Native app. Expo also handles a lot of configuration automatically, making it easier to manage multimedia, handle notifications, and compile the app.
- **React Navigation**: This library helps with navigating between different screens within the app smoothly and efficiently.
- **AsyncStorage from React Native**: Used for local storage of user data like tokens and session states across app restarts and reloads.

2. Backend API

- **FastAPI**: A modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. The key features that make FastAPI a great choice for this kind of project include its speed, ease of use, and robustness.
- **Uvicorn**: An ASGI server for Python, used to run the FastAPI application. It helps in handling asynchronous requests and makes the application scalable and fast.

3. Database and Authentication

- **MongoDB Atlas**: A cloud database service used for storing all persistent data including user credentials, menu items, and order details. It offers high performance, scalability, and flexibility, making it a suitable choice for applications needing to handle large volumes of data.
- **JWT (JSON Web Tokens)**: Used for securing the backend communication and effectively managing user authentication and sessions. It ensures that each request to the server is authenticated and authorized properly.

4. Additional Python Packages

- **Pymongo**: A Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.
- **Passlib & python-jose**: These libraries are used for password hashing and JWT operations respectively, which are critical for secure authentication mechanisms.
- **Python-dotenv**: Used for loading environment variables from a `.env` file which is essential for managing configuration options and secret keys securely outside of the main codebase.

Configuration and Integration

- The mobile app makes requests to the FastAPI backend, handling tasks such as user authentication (login/register), menu browsing, and order processing.
- Data is stored and retrieved from MongoDB Atlas, ensuring that all interactions are persistent and stateful.
- Security is a top priority with hashed passwords stored in the database and JWT used for session management to prevent unauthorized access.

This architecture not only ensures that the application is robust, secure, and scalable but also provides a seamless user experience whether on a web browser or on a mobile device. The use of modern frameworks and technologies like React Native, FastAPI, and MongoDB Atlas highlights the application's commitment to using cutting-edge technology to improve user satisfaction and operational efficiency.

Features

- **User Authentication**: Secure login and registration system.
- **Intuitive Interface**: Simple and user-friendly interface for ordering.
- **Shopping Cart Functionality**: Add items to your cart and manage them effortlessly.
- **Order Checkout**: Review your selections and prices before finalizing the order.

Getting Started

To get a local copy up and running follow these simple steps.

Prerequisites

- npm

  ```
  npm install npm@latest -g
  ```

- Expo CLI

  ```
  npm install -g expo-cli
  ```

Installation

1. Clone the repository.

   ```
   git clone https://github.com/AgentMrCow/Restaurant-Ordering-Mobile-App.git
   ```

2. Install NPM packages.

   ```
   npm install
   ```

3. Start the application.

   ```
   expo start
   ```

For the backend service to handle user authentication and order processing:

1. Ensure you have Python installed on your system.
2. Install the necessary Python packages using `pip` (or `pip3` if Python 3 is not your default Python version):

   ```
   pip install fastapi uvicorn pymongo python-multipart python-jose passlib python-dotenv
   ```

3. Add Required Variables: Create a new `.env` file and add the following lines:

```
DATABASE_URI="your_mongodb_connection_string_here"
JWT_SECRET_KEY="your_secret_key_here"
```

- Replace `your_mongodb_connection_string_here` with your actual MongoDB connection string.
- Replace `your_secret_key_here` with a strong secret key for JWT authentication.

Example Content for `.env` File:

```
DATABASE_URI="mongodb+srv://username:password@cluster0.example.mongodb.net/"
JWT_SECRET_KEY="thisisasecretkey"
```

4. Run the FastAPI server:

```
uvicorn main:app --reload
```

Appendix: Source Code

React Native Frontend Code

```
// App.js
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import DateTimePickerModal from 'react-native-modal-datetime-picker';
import { StatusBar } from "expo-status-bar";
import React, { useEffect, createContext, useState, useContext } from "react";
import {
  StyleSheet,
  Text,
  View,
  Image,
  TextInput,
  Button,
  TouchableOpacity,
  ScrollView,
} from "react-native";
import AsyncStorage from '@react-native-async-storage/async-storage';
import { decode as atob } from 'base-64';
import { Card } from 'react-native-paper';
import { useFocusEffect } from '@react-navigation/native';

const CartContext = createContext();

export const useCart = () => useContext(CartContext);

export const CartProvider = ({ children }) => {
  const [cart, setCart] = useState([]);

  const addToCart = (item) => {
    setCart(currentCart => {
      const index = currentCart.findIndex(i => i.name === item.name);
```

```
      if (index > -1) {
        const newCart = [...currentCart];
        newCart[index].quantity += 1;
        return newCart;
      } else {
        return [...currentCart, { ...item, quantity: 1 }];
      }
    });
  };

  const removeFromCart = (itemToRemove) => {
    setCart(currentCart => {
      const index = currentCart.findIndex(i => i.name === itemToRemove.name);
      if (index > -1 && currentCart[index].quantity > 1) {
        const newCart = [...currentCart];
        newCart[index].quantity -= 1;
        return newCart;
      } else {
        return currentCart.filter(item => item.name !== itemToRemove.name);
      }
    });
  };

  const clearCart = () => {
    setCart([]);
  };

  return (
    <CartContext.Provider value={{ cart, addToCart, removeFromCart, clearCart }}>
      {children}
    </CartContext.Provider>
  );
};


const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <CartProvider>
      <NavigationContainer>
        <Stack.Navigator>
          <Stack.Screen name="Home" component={HomeScreen} />
          <Stack.Screen name="Register" component={RegisterScreen} />
          <Stack.Screen name="Store" component={StoreScreen} />
          <Stack.Screen name="Cart" component={CartScreen} />
        </Stack.Navigator>
      </NavigationContainer>
    </CartProvider>
  );
}
```

```
function getEmailFromToken(token) {
  try {
    const payload = JSON.parse(atob(token.split('.')[1]));
    return payload.sub;
  } catch (e) {
    console.error('Failed to decode token:', e);
    return null;
  }
}

const HomeScreen = ({ navigation }) => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [emailError, setEmailError] = useState("")
  const [passwordError, setPasswordError] = useState("")

  useEffect(() => {
    checkLoginStatus();
  }, []);

  const checkLoginStatus = async () => {
    const token = await AsyncStorage.getItem('userToken');
    if (token) {
      alert('You have already logged in.');
      navigation.navigate("Store");
    }
  }

  function onLoginPressed() {
    checkLoginStatus();
    // Resetting error states
    setEmailError("");
    setPasswordError("");

    let valid = true;

    // Basic validation for email and password
    if (email === "") {
      setEmailError("Please enter your email");
      valid = false;
      console.log("Email validation failed: No email provided.");
    } else if (!/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/.test(email)) {
      setEmailError("Please enter a valid email");
      valid = false;
      console.log("Email validation failed: Invalid email format.");
    }

    if (password === "") {
      setPasswordError("Please enter a password");
      valid = false;
      console.log("Password validation failed: No password provided.");
    } else if (password.length < 8) {
```

```
      setPasswordError("The password must be 8 characters or longer");
      valid = false;
      console.log("Password validation failed: Password is too short.");
    }

    // If validation passes, proceed to make a login request
    if (valid) {
      console.log("Validation passed, attempting to log in.");
      fetch("http://10.0.2.2:8000/auth", {
        method: "POST",
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ email, password })
      })
        .then(r => r.json())
        .then(r => {
          if ('success' === r.message) {
            console.log("Login successful.");
            // Store the token received upon successful login
            const token = r.token; // Assuming the token is returned in the response
            console.log("Token received:", token);

            // Verify the token by calling the `/verify` endpoint
            console.log("Attempting to verify token.");
            fetch("http://10.0.2.2:8000/verify", {
              method: "POST",
              headers: {
                'Content-Type': 'application/json',
                'jwt-token': token // Sending the token in the request headers
              },
            })
              .then(verifyResponse => verifyResponse.json())
              .then(async (verifyResult) => {
                if (verifyResult.status === 'logged in') {
                  console.log("Token verification successful.");
                  // If the token is verified successfully, navigate to the Store
screen
                  console.log("Login successful, token:", r.token);
                  try {
                    await AsyncStorage.setItem('userToken', r.token);
                    navigation.navigate("Store");
                  } catch (error) {
                    console.error("AsyncStorage error:", error);
                  }
                } else {
                  console.log("Token verification failed with status:",
verifyResult.status);
                }
              })
              .catch(verifyError => console.error("Verification error:",
verifyError));
```

```
        } else if (r.message === 'fail') {
          setEmailError('Login failed. Please check your credentials.');
        } else {
          console.log(r)
          console.log("Login failed with message:", r.message);
        }
      })
      .catch(loginError => console.error("Login error:", loginError));
  } else {
    console.log("Validation failed, not attempting to log in.");
  }
}

const onRegisterPressed = async () => {
  const token = await AsyncStorage.getItem('userToken');
  if (token) {
    alert('You have already register.');
    navigation.navigate("Store");
  } else {
    navigation.navigate("Register");
  }
}


return (
  <View style={styles.container}>
    <Text style={styles.storeTitle}>IERG3842 Mobile App</Text>
    <Text style={styles.describe}>Chinese Restaurant Online Ordering System</Text>
    <Text style={styles.userInfo}>By Niu Ka Ngai</Text>
    <Text style={styles.userInfo}>Student ID: 1155174712</Text>
    <Image style={styles.image} source={require("./assets/food.png")} />
    <StatusBar style="auto" />

    <View style={styles.inputWithErrorView}>
      <View style={styles.inputView}>
        <TextInput
          style={styles.TextInput}
          placeholder="Email"
          placeholderTextColor="#003f5c"
          onChangeText={(email) => setEmail(email)}
        />
      </View>
      {emailError !== "" && <Text style={styles.errorText}>{emailError}</Text>}
    </View>

    <View style={styles.inputWithErrorView}>
      <View style={styles.inputView}>
        <TextInput
          style={styles.TextInput}
          placeholder="Password"
          placeholderTextColor="#003f5c"
          secureTextEntry={true}
```

```
            onChangeText={(password) => setPassword(password)}
          />
        </View>
        {passwordError !== "" && <Text style={styles.errorText}>{passwordError}
</Text>}
      </View>

      <TouchableOpacity style={styles.loginBtn} onPress={onLoginPressed}>
        <Text style={styles.loginText}>Login</Text>
      </TouchableOpacity>
      <TouchableOpacity style={styles.loginBtn} onPress={onRegisterPressed}>
        <Text style={styles.loginText}>Register</Text>
      </TouchableOpacity>
    </View>
  );
};

const RegisterScreen = ({ navigation }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [dateOfBirth, setDateOfBirth] = useState(null);
  const [isDatePickerVisible, setDatePickerVisibility] = useState(false);
  const [gender, setGender] = useState('male');
  const [emailError, setEmailError] = useState('');
  const [passwordError, setPasswordError] = useState('');
  const [confirmPasswordError, setConfirmPasswordError] = useState('');
  const [dobError, setDobError] = useState("");

  const showDatePicker = () => {
    setDatePickerVisibility(true);
  };

  const hideDatePicker = () => {
    setDatePickerVisibility(false);
  };

  const handleConfirmDate = (date) => {
    setDateOfBirth(date);
    hideDatePicker();
  };

  const validateFields = () => {
    let isValid = true;
    setEmailError('');
    setPasswordError('');
    setConfirmPasswordError('');
    setDobError('');

    if (email === '') {
      setEmailError("Please enter your email");
      isValid = false;
```

```
  } else if (!/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/.test(email)) {
    setEmailError("Please enter a valid email");
    isValid = false;
  }

  if (password === '') {
    setPasswordError("Please enter your password");
    isValid = false;
  } else if (password.length < 8) {
    setPasswordError("The password must be 8 characters or longer");
    isValid = false;
  }

  if (confirmPassword === '') {
    setConfirmPasswordError("Please enter your confirm password");
    isValid = false;
  } else if (confirmPassword !== password) {
    setConfirmPasswordError("Passwords don't match!");
    isValid = false;
  }

  if (!dateOfBirth) {
    setDobError("Please select your date of birth");
    isValid = false;
  } else if (1) {
    // xxx
  }

  return isValid;
};

const handleRegister = () => {
  if (validateFields()) {
    // Step 1: Invoke the check-account endpoint
    fetch("http://10.0.2.2:8000/check-account", {
      method: "POST",
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ email })
    })
      .then(response => response.json())
      .then(data => {
        // Step 2: Process the response
        if (data.userExists) {
          // If user exists, set an email error
          setEmailError("An account with this email already exists.");
        } else {
          // Step 3: Proceed with registration if user does not exist
          console.log("Proceeding with registration for:", email);
          fetch("http://10.0.2.2:8000/register", {
            method: "POST",
```

```
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({ email, password, confirmPassword, dateOfBirth,
gender })
          })
            .then(registerResponse => registerResponse.json())
            .then(async (data) => {
              if (data.message === 'User registered successfully') {
                console.log("Register successful. Token:", data.token);
                try {
                  await AsyncStorage.setItem('userToken', data.token);
                  navigation.navigate("Store");
                } catch (error) {
                  console.error("AsyncStorage error:", error);
                }
              } else {
                setEmailError(data.message || "Registration failed. Please try
again.");
              }
            })
            .catch(error => {
              console.error("Registration error:", error);
              setEmailError("An error occurred during registration. Please try
again.");
            });
          }
        })
        .catch(error => {
          console.error("Error checking account:", error);
          setEmailError("An error occurred. Please try again later.");
        });
    }
  };


  return (
    <ScrollView contentContainerStyle={styles.registerContainer}>
      <Text style={styles.title}>Register</Text>

      <View style={styles.inputContainer}>
        <TextInput
          style={styles.textInput}
          placeholder="Email"
          onChangeText={setEmail}
          value={email}
        />
        {emailError !== "" && <Text style={styles.errorText}>{emailError}</Text>}
      </View>

      <View style={styles.inputContainer}>
        <TextInput
```

```
          style={styles.textInput}
          placeholder="Password"
          secureTextEntry
          onChangeText={setPassword}
          value={password}
        />
        {passwordError !== "" && <Text style={styles.errorText}>{passwordError}
</Text>}
      </View>

      <View style={styles.inputContainer}>
        <TextInput
          style={styles.textInput}
          placeholder="Confirm Password"
          secureTextEntry
          onChangeText={setConfirmPassword}
          value={confirmPassword}
        />
        {confirmPasswordError !== "" && <Text style={styles.errorText}>
{confirmPasswordError}</Text>}
      </View>

      <TouchableOpacity style={styles.datePickerButton} onPress={showDatePicker}>
        <Text style={styles.datePickerText}>
          {`Date of Birth: ${dateOfBirth ? dateOfBirth.toLocaleDateString() : "Please
Select Date"}`}
        </Text>
      </TouchableOpacity>
      {dobError !== "" && <Text style={styles.errorText}>{dobError}</Text>}

      <DateTimePickerModal
        isVisible={isDatePickerVisible}
        mode="date"
        maximumDate={new Date()}
        onConfirm={handleConfirmDate}
        onCancel={hideDatePicker}
      />
      <View style={styles.genderContainer}>
        <TouchableOpacity
          style={gender === 'male' ? styles.genderButtonSelected :
styles.genderButton}
          onPress={() => setGender('male')}
        >
          <Text style={styles.genderButtonText}>Male</Text>
        </TouchableOpacity>
        <TouchableOpacity
          style={gender === 'female' ? styles.genderButtonSelected :
styles.genderButton}
          onPress={() => setGender('female')}
        >
          <Text style={styles.genderButtonText}>Female</Text>
        </TouchableOpacity>
```

```
      </View>

      <TouchableOpacity style={styles.registerButton} onPress={handleRegister}>
        <Text style={styles.registerButtonText}>Register</Text>
      </TouchableOpacity>
    </ScrollView>
  );
};

const StoreScreen = ({ navigation }) => {
  const [items, setItems] = useState([]);
  const { cart, addToCart } = useContext(CartContext);

  useEffect(() => {
    fetch('http://10.0.2.2:8000/items')
      .then(response => response.json())
      .then(data => setItems(data))
      .catch(error => console.error('Error fetching items:', error));
  }, []);

  return (
    <View style={styles.storeContainer}>
      <View style={styles.navigationBar}>
        <Text style={styles.navItem}>Appetizer</Text>
        <Text style={styles.navItemActive}>Main Course</Text>
        <Text style={styles.navItem}>Dessert</Text>
        <Text style={styles.navItem}>Beverage</Text>
      </View>
      <ScrollView style={styles.storeScreen}>
        <View style={styles.menuBar}>
          <TouchableOpacity
            style={[styles.menuButton, cart.length === 0 &&
styles.menuButtonDisabled]}
            onPress={() => cart.length > 0 && navigation.navigate('Cart')}
            disabled={cart.length === 0}>
            <Image source={require('./assets/shopping-cart.png')} style={styles.icon}
/>
            <Text style={styles.menuText}>Cart ({cart.length})</Text>
          </TouchableOpacity>
          <TouchableOpacity style={styles.menuButton} onPress={() => {
            AsyncStorage.removeItem('userToken');
            navigation.navigate("Home");
          }}>
            <Image source={require('./assets/log-out.512x512.png')} style=
{styles.icon} />
            <Text style={styles.menuText}>Logout</Text>
          </TouchableOpacity>
        </View>
        {items.map((item, index) => (
          <Card key={index}>
            <Card.Title title={item.name} subtitle={`$${item.price}`} />
            <Card.Cover source={{ uri: item.image_url }} style={styles.cardImage} />
```

```
            <Card.Actions>
              <Button onPress={() => addToCart(item)} title="Add to Cart"
color="#ac2925" />
            </Card.Actions>
          </Card>
        ))}
      </ScrollView>
    </View>
  );
};



const CartScreen = ({ navigation }) => {
  const { cart, removeFromCart, clearCart } = useContext(CartContext);

  const total_price = cart.reduce((acc, item) => acc + (item.price * item.quantity),
0);

  const handleCheckout = async () => {
    const token = await AsyncStorage.getItem('userToken');
    const userEmail = getEmailFromToken(token);
    const items = cart.map(item => ({ name: item.name, quantity: item.quantity }));

    fetch('http://10.0.2.2:8000/purchase', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email: userEmail, items, total_price })
    })
      .then(response => response.json())
      .then(data => {
        alert("Purchase successful!");
        clearCart();
        navigation.navigate('Store');
      })
      .catch(error => {
        console.error('Checkout error:', error);
        alert("Error processing your purchase.");
      });
  };

  return (
    <ScrollView contentContainerStyle={styles.cartContainer}>
      {cart.map((item, index) => (
        <View key={index} style={styles.cartItem}>
          <Image source={{ uri: item.image_url }} style={styles.cartItemImage} />
          <View style={styles.cartItemDetail}>
            <Text style={styles.itemName}>{item.name} - ${item.price} x
{item.quantity} (Subtotal: ${item.price * item.quantity})</Text>
            <TouchableOpacity onPress={() => removeFromCart(item)} style=
{styles.removeItemBtn}>
              <Text style={styles.removeItemText}>Remove</Text>
```

```
            </TouchableOpacity>
          </View>
        </View>
      ))}
      <Text style={styles.totalPriceText}>Total: ${total_price.toFixed(2)}</Text>
      <Button title="Continue Ordering" onPress={() => navigation.goBack()} />
      <View style={styles.space}></View>
      <Button title="Checkout" onPress={handleCheckout} color="#007bff" />
    </ScrollView>
  );
};




const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#f5f5f5",
    alignItems: "center",
    justifyContent: "center",
    padding: 20,
  },
  storeTitle: {
    fontSize: 24,
    fontWeight: "bold",
    color: "#007bff",
    marginBottom: 10,
  },
  describe: {
    fontSize: 16,
    color: "#ac2925",
    marginBottom: 5,
    fontWeight: "bold",
  },
  userInfo: {
    fontSize: 16,
    color: "#008080",
    marginBottom: 5,
    fontWeight: "bold",
  },
  image: {
    width: 400,
    height: 200,
  },
  inputWithErrorView: {
    width: "100%",
    marginBottom: 15,
  },
  inputView: {
    width: "100%",
    backgroundColor: "#fff",
```

```
    borderRadius: 20,
    padding: 15,
    marginBottom: 15,
    borderWidth: 1,
    borderColor: "#ddd",
  },
  TextInput: {
    height: 40,
    fontSize: 16,
  },
  loginBtn: {
    width: "100%",
    borderRadius: 20,
    height: 50,
    alignItems: "center",
    justifyContent: "center",
    marginTop: 20,
    backgroundColor: "#007bff",
    elevation: 5,
  },
  loginText: {
    color: "#fff",
    fontSize: 18,
    fontWeight: "bold",
  },

  // error text define below
  // Register
  registerContainer: {
    flexGrow: 1,
    justifyContent: 'center',
    padding: 20,
  },
  inputContainer: {
    marginBottom: 15,
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },
  textInput: {
    width: '100%',
    height: 50,
    borderColor: '#ccc',
    borderWidth: 1,
    borderRadius: 5,
    padding: 10,
    backgroundColor: '#f9f9f9',
  },
  datePickerButton: {
```

```
    width: '100%',
    padding: 15,
    backgroundColor: '#f0f0f0',
    borderRadius: 5,
    marginBottom: 15,
    alignItems: 'center',
  },
  datePickerText: {
    fontSize: 16,
  },
  genderContainer: {
    flexDirection: 'row',
    justifyContent: 'center',
    marginBottom: 20,
  },
  genderButton: {
    padding: 10,
    flex: 1,
    alignItems: 'center',
    backgroundColor: '#e7e7e7',
    borderRadius: 5,
    marginHorizontal: 0, //5
  },
  genderButtonSelected: {
    padding: 10,
    flex: 1,
    alignItems: 'center',
    backgroundColor: '#007bff',
    borderRadius: 5,
    marginHorizontal: 0, // 5
  },
  genderButtonText: {
    color: '#fff',
    fontWeight: 'bold',
  },
  registerButton: {
    width: '100%',
    padding: 15,
    backgroundColor: '#007bff',
    borderRadius: 5,
    alignItems: 'center',
    marginBottom: 20,
  },
  registerButtonText: {
    color: '#fff',
    fontSize: 16,
    fontWeight: 'bold',
  },
  datePickerButton: {
    width: '100%',
    padding: 15,
    backgroundColor: '#e8e8e8',
```

```
    borderRadius: 8,
    marginBottom: 10,
    alignItems: 'center',
    justifyContent: 'center',
    borderWidth: 1,
    borderColor: '#d0d0d0',
  },
  datePickerText: {
    fontSize: 16,
    color: '#333',
  },
  errorText: {
    color: 'red',
    textAlign: 'left',
    marginBottom: 5,
  },

  // store
  storeContainer: {
    flex: 1,
    paddingTop: 10,
  },
  navigationBar: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    marginBottom: 10,
  },
  navItem: {
    fontWeight: 'bold',
    color: '#666',
  },
  navItemActive: {
    fontWeight: 'bold',
    color: '#007bff',
  },
  storeScreen: {
    flex: 1,
    backgroundColor: "#f5f5f5",
  },
  menuBar: {
    flexDirection: 'row',
    justifyContent: 'space-evenly',
    marginVertical: 10,
  },
  menuButton: {
    flexDirection: 'row',
    backgroundColor: "#007bff",
    padding: 10,
    borderRadius: 5,
    alignItems: 'center',
    justifyContent: 'center',
    elevation: 3,
```

```
},
menuButtonDisabled: {
  backgroundColor: "#ccc",
},
icon: {
  width: 20,
  height: 20,
  marginRight: 10,
},
menuText: {
  fontSize: 16,
  color: '#ffffff',
  fontWeight: 'bold',
},
cardImage: {
  width: '100%',
  height: 150,
},

// cart
cartContainer: {
  padding: 20,
  backgroundColor: "#fff",
},
cartItem: {
  flexDirection: 'row',
  marginBottom: 10,
  borderBottomWidth: 1,
  borderBottomColor: '#ccc',
  paddingBottom: 10,
},
cartItemImage: {
  width: 100,
  height: 100,
  marginRight: 10,
},
cartItemDetail: {
  flex: 1,
  justifyContent: 'space-between',
},
itemName: {
  fontSize: 16,
  fontWeight: 'bold',
},
removeItemBtn: {
  padding: 10,
  backgroundColor: '#ff4444',
  borderRadius: 5,
},
removeItemText: {
  color: '#fff',
  fontSize: 14,
```

```
    textAlign: 'center',
  },
  totalPriceText: {
    fontSize: 18,
    fontWeight: 'bold',
    color: '#000',
    paddingVertical: 10
  },
  space: {
    height: 20,
  }
});
```

FastAPI Backend Code

```python
# main.py
from fastapi import FastAPI, Header, Depends, APIRouter, HTTPException
from fastapi.responses import JSONResponse
from fastapi.middleware.cors import CORSMiddleware
from pymongo import MongoClient
from pymongo.server_api import ServerApi
from jose import jwt, JWTError
from passlib.context import CryptContext
import uvicorn
from pydantic import BaseModel, EmailStr, validator, ValidationError
from datetime import datetime
from typing import List
import os
from bson import ObjectId
from fastapi.encoders import jsonable_encoder
from dotenv.main import load_dotenv

# Take environment variables from .env file.
load_dotenv()

DATABASE_URI = os.getenv("DATABASE_URI")
JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY")

if not DATABASE_URI:
    raise ValueError("DATABASE_URI is not set in .env file")

if not JWT_SECRET_KEY:
    raise ValueError("JWT_SECRET_KEY is not set in .env file")

# Configuration and Constants
client = MongoClient(DATABASE_URI, server_api=ServerApi('1'))
db = client.myStore

# Test MongoDB connection
try:
    client.admin.command('ping')
```

```
    print("Successfully connected to MongoDB!")
except Exception as e:
    print(f"Failed to connect to MongoDB: {e}")

# FastAPI and Security Setup
app = FastAPI()
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# Pydantic Models for Request Data
class AuthData(BaseModel):
    email: EmailStr
    password: str

    # @validator('password')
    # def password_length(cls, value):
    #     if len(value) < 8:
    #         raise ValueError('Password must be at least 8 characters')
    #     return value

class RegistrationData(AuthData):
    confirmPassword: str
    dateOfBirth: datetime
    gender: str

    # @validator('confirmPassword')
    # def passwords_match(cls, v, values, **kwargs):
    #     if 'password' in values and v != values['password']:
    #         raise ValueError('Passwords do not match')
    #     return v

    # @validator('dateOfBirth')
    # def dob_must_be_in_the_past(cls, v):
    #     if v >= datetime.now():
    #         raise ValueError('Date of Birth must be in the past')
    #     return v

class EmailData(BaseModel):
    email: EmailStr

router = APIRouter()

class Item(BaseModel):
    name: str
    description: str = None
    price: float
    image_url: str = None
    quantity: int = 1

class PurchaseItem(BaseModel):
    name: str
    quantity: int
```

```python
class Purchase(BaseModel):
    email: EmailStr
    items: List[PurchaseItem]
    total_price: float

# Helper function to handle MongoDB ObjectId
def object_id_handler(obj):
    if isinstance(obj, ObjectId):
        return str(obj)
    raise TypeError

@router.get("/items")
async def get_items():
    items = list(db.items.find({}, {'_id': 0}))
    return jsonable_encoder(items, custom_encoder={ObjectId: object_id_handler})

@router.post("/items")
async def add_item(item: Item):
    db.items.insert_one(item.dict())
    return {"message": "Item added successfully"}

@app.post("/purchase")
async def make_purchase(purchase: Purchase):
    # Example logic to process each item by quantity
    for item in purchase.items:
        # Process item.name with item.quantity
        pass
    db.purchases.insert_one(purchase.dict())
    return {"message": "Purchase recorded successfully"}

app.include_router(router)

# Enable CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# Routes
@app.get("/")
async def root():
    return {"message": "Auth API. Please use POST /auth & POST /verify for
authentication"}

@app.post("/auth")
async def authenticate_user(auth_data: AuthData):
    user = db.userInfo.find_one({"email": auth_data.email})
    if user and pwd_context.verify(auth_data.password, user["password"]):
        token_data = {"sub": auth_data.email}
        token = jwt.encode(token_data, JWT_SECRET_KEY, algorithm="HS256")
```

```
        return {"message": "success", "token": token}
    else:
        print("Authentication failed")
        return {"message": "fail"}, 401

@app.post("/verify")
async def verify_token(token: str = Header(None, alias='jwt-token')):
    if not token:
        return JSONResponse(status_code=400, content={"message": "Token missing"})
    try:
        payload = jwt.decode(token, JWT_SECRET_KEY, algorithms=["HS256"])
        return {"status": "logged in", "message": "success"}
    except JWTError:
        return JSONResponse(status_code=401, content={"status": "invalid auth",
"message": "error"})


@app.post("/check-account")
async def check_account(email_data: EmailData):
    try:
        user_exists = db.userInfo.find_one({"email": email_data.email}) is not None
        return {"userExists": user_exists}
    except Exception as e:
        print(f"Error in check-account: {str(e)}")
        raise HTTPException(status_code=400, detail=str(e))


@app.post("/register")
async def register_user(reg_data: RegistrationData):
    try:
        if reg_data.password != reg_data.confirmPassword:
            return JSONResponse(status_code=400, content={"message": "Passwords do not
match"})
        if db.userInfo.find_one({"email": reg_data.email}):
            return JSONResponse(status_code=400, content={"message": "Email already
registered"})
        hashed_password = pwd_context.hash(reg_data.password)
        db.userInfo.insert_one({
            "email": reg_data.email,
            "password": hashed_password,
            "dateOfBirth": reg_data.dateOfBirth,
            "gender": reg_data.gender
        })
        token_data = {"sub": reg_data.email}
        token = jwt.encode(token_data, JWT_SECRET_KEY, algorithm="HS256")
        return {"message": "User registered successfully", "token": token}
    except Exception as e:
        print(f"Error in register: {str(e)}")
        raise HTTPException(status_code=400, detail=str(e))
```

```
if __name__ == "__main__":
    uvicorn.run("filename:app", host="0.0.0.0", port=8000, reload=True, debug=True)
```