Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering (FSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Weather API Integration and Multiplayer Game Using Sockets

Course Title: Computer Networking Lab
Course Code: CSE-312
Section: 222-D3

Students Details

Name	ID
Md. Zehadul Islam	222902069
Md. Abdullah Al Moin	222902070

Submission Date: 24.12.2024
Course Teacher's Name: Maisha Muntaha

[For teachers use only: Don't write anything inside this box]

Project Report Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

Weather Application: This project uses networking to fetch weather data from an online server using an API. Users can input a city name, and the application will display weather details like temperature and humidity. Multiplayer Tic-Tac-Toe Game: This project allows two players to play Tic-Tac-Toe over a network. It uses client-server communication so players can connect remotely, make moves, and see the game updates live. [1] [2] [3]

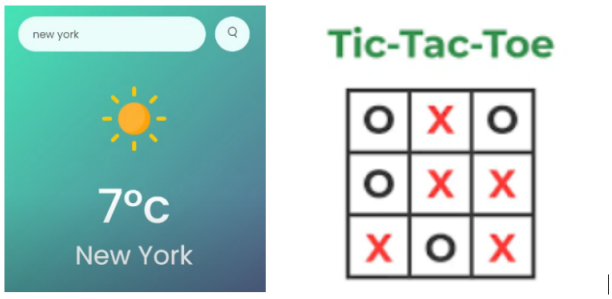

Figure 1.1: Weather API and Multiplayer Game Using Sockets

1.2 Motivation

The Weather Application helps us learn how to use networking to fetch real-time data and create useful tools for users. The Multiplayer Tic-Tac-Toe Game project teaches us about client-server communication and how to make interactive games that can be played online, building our networking and programming skills.

1.2.1 Problem Statement

For the Weather Application, the main task is to fetch accurate weather data from an online API based on the user's city and display it clearly. For the Tic-Tac-Toe Game, the goal is to connect two players online, allowing them to make moves and see updates instantly. The challenge is keeping both players' moves synchronized in real-time without errors.

1.2.2 Complex Engineering Problem

The following table summarizes the attributes related to the complex engineering problem addressed by the project:

Table 1.1: Summary of the attributes touched by the mentioned project

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	Basic knowledge of networking, using APIs, and socket programming is needed. Understanding of Java will help to build the applications.
P2: Range of conflicting requirements	Balancing between real-time updates and network stability, handling API response delays,and ensuring the app works on all devices.
P3: Depth of analysis required	API responses, socket communication, and po- tential network issues is needed for smooth,error-free performance.
P4: Familiarity of issues	Common issues include API errors, network connection problems, and handling synchronization between two players in the game.
P5: Extent of applicable codes	API integration, socket programming, user interface design, and handling network errors and real-time updates.
P6: Extent of stakeholder involvement and conflicting requirements	Stakeholders include users and developers, balancing user experience with network performance needs.
P7: Interdependence	The Weather App depends on the API for data, while the Tic-Tac-Toe Game relies on socket communication for real-time updates. Both projects need a stable network connection for smooth performance.

1.3 Design Goals/Objectives

The goals and objectives of the project are as follows:

1. **User-friendly Interface:** Create a simple and clear interface for both the Weather-App and Tic-Tac-Toe Game.
2. **API Integration:** Fetch real-time weather data using a reliable weather API.
3. **Stable Connection:** Use sockets for smooth, real-time communication in the Tic-Tac-Toe game.
4. **Error Handling:** Handle network issues and invalid user input gracefully.
5. **Cross-Platform Support:** Ensure the application works on different devices with-out issues.

1.4 Application

The solutions to the Traveling Knight Problem, Train Swapping Problem, and Travel in Desert Problem offer diverse and efficient applications:

1. **Weather App:** Users can input any city name to get up-to-date weather information like temperature, humidity, and weather conditions. The app uses an API to gather real-time data and show it in a simple, easy-to-read format.
2. **Multiplayer Tic-Tac-Toe Game:** Two players can connect to play Tic-Tac-Toe over a network using socket programming. The game updates the moves of both players in real-time, allowing them to compete remotely with clear and instant feedback.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This project, "Bayath's OP Integration with Mainstream Games Using Sockets," demonstrates the use of networking and game logic for creating a multiplayer Tic-Tac-Toe game and a Weather API integration. The project utilizes Java for implementing socket programming, creating a GUI-based Tic-Tac-Toe game, and fetching weather data using an API. [4] [**?**].

2.2 Project Details

The project integrates two main functionalities:

1. A Weather API application that fetches and displays weather information for a given city.
2. A multiplayer Tic-Tac-Toe game built using client-server architecture.

Both applications leverage Java's networking and GUI capabilities to provide interactive user experiences.

2.2.1 Weather API Application

- **Input:** A city name entered by the user.
- **Process:** The application makes a GET request to the OpenWeatherMap API and parses the JSON response to extract weather details.
- **Output:** Displays the weather description and temperature in Celsius.

2.2.2 Multiplayer Tic-Tac-Toe Game

The program calculates BMI using a simple formula: BMI = Weight / Height² It checks the result and tells if the user is:

- **Input:** Players' moves on a 3x3 grid.
- **Process:** A server manages the game state, validates moves, and determines the winner.
- **Output:** Updates the game board and displays the winner or prompts for invalid moves.

2.3 Implementation

2.3.1 Workflow

The workflow of the project involves the following steps:

1. **Weather API Application:**
    - Start the application and display the GUI.
    - User inputs a city name.
    - Fetch weather data from the API and display results.

1. **Tic-Tac-Toe Game:**
    - Server initializes the game board and waits for connections.
    - Two clients connect to the server.
    - Players take turns making moves.
    - The server validates moves, updates the board, and checks for a winner.
    - The game ends when a player wins or all positions are filled.

2.3.2 Tools and Libraries

1. **Java:** For implementing the logic, GUI, and networking.
2. **Socket Programming:** Used for real-time communication between the server and clients in the Tic-Tac-Toe game.
3. **Swing:** For creating graphical user interfaces.
4. **OpenWeatherMap API:** Provides weather data for the Weather API application.

2.3.3 Implementation Details

1. **Weather API Application:**

   - **Input Handling:** Accepts a city name through a GUI text field.
   - **API Request:** Sends a GET request to the OpenWeatherMap API using Java's HttpURLConnection.
   - **Data Parsing:**Extracts weather details from the JSON response.
   - **Output Handling:** Displays the weather description and temperature in a text area.

1. **Tic-Tac-Toe Game:**

   - **Server:**
     - Manages the game state and validates moves .
     - Sends updates to connected clients.
     - Determines the winner or declares a draw.
   - **Client:**
     - Displays the game board using a 3x3 grid of buttons.
     - Allows players to make moves.
     - Updates the board based on server responses.

2.4 Algorithms

Weather API Application:

(a) Display the application GUI.

(b) Prompt the user to input a city name.

(c) Fetch weather data using the OpenWeatherMap API.

(d) Parse the JSON response to extract weather and temperature details.

(e) Display the weather information in the GUI.

Tic-Tac-Toe Game:

(a) Server initializes the game board and waits for client connections.

(b) Players alternate making moves:

- Validate the move.
- Update the board.
- Check for a winner or draw.

(c) Notify clients about the game state after each move.

(d) End the game when there is a winner or draw.

2.5 Flow-Chart

- **I added a Flow-chart to show how the program works.**

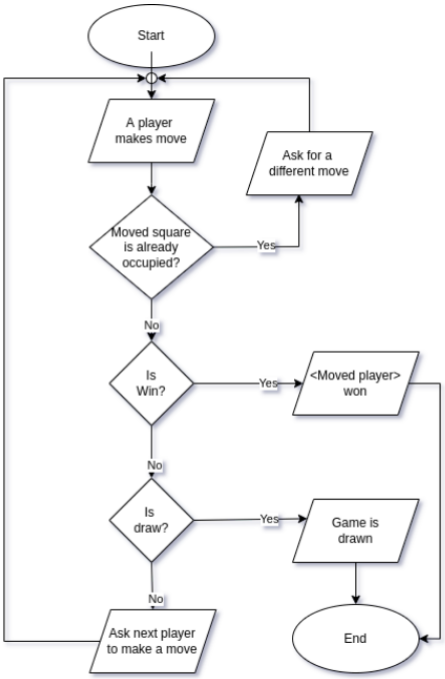

Figure 2.1: Flow-Chart For Tic-tac-toe

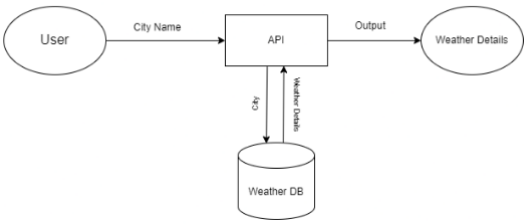

Figure 2.2: Flow-Chart For Wheather API

2.6 Code Implementation

The code for the project is divided into several modules:

(a) **Weather API Application:**

- Handles the GUI creation using Swing.

```
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import javax.swing.*;
import java.awt.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

public class WeatherApp {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new WeatherApp().createAndShowGUI());
    }

    private void createAndShowGUI() {
        JFrame frame = new JFrame(title: "Weather App");
        frame.setDefaultCloseOperation(operation:JFrame.EXIT_ON_CLOSE);
        frame.setSize(width: 400, height:300);

        JLabel cityLabel = new JLabel(text: "City:");
        JTextField cityField = new JTextField(columns: 15);
        JButton getWeatherButton = new JButton(text: "Get Weather");
        JTextArea weatherInfo = new JTextArea(rows: 10, columns: 30);
        weatherInfo.setEditable(b: false);
```

Figure 2.3: GUI creation using Swing

- Fetches and displays weather data:

```
        getWeatherButton.addActionListener(e -> {
            String city = cityField.getText();
            if (city.trim().isEmpty()) {
                weatherInfo.setText(t: "Please enter a city name.");
            } else {
                String weatherData = getWeather(city);
                weatherInfo.setText(t: weatherData);
            }
        });
    }

    private String getWeather(String city) {
        String apiKey = "b11d429473ec5bee2531e4bccf8276d3"; // Replace with your API key
        try {
            String encodedCity = URLEncoder.encode(s: city, enc:"UTF-8");
            String urlString = "https://api.openweathermap.org/data/2.5/weather?q=" + encodedCity + "&appid=" + apiKey

            URL url = new URL(spec: urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod(method:"GET");

            int responseCode = conn.getResponseCode();
            if (responseCode != 200) {
                return "Error: Received HTTP code " + responseCode;
            }

            BufferedReader in = new BufferedReader(new InputStreamReader(in: conn.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;
```

Figure 2.4: Fetches and displays weather data

(b) **Tic-Tac-Toe Game :**

- Server:

```
public class TicTacToeServer {
    private static final int PORT = 12345;
    private static ServerSocket serverSocket;
    private static ArrayList<PlayerHandler> players = new ArrayList<>();
    private static char[] board = new char[9];
    private static int currentPlayer = 0;

    public static void main(String[] args) {
        try {
            serverSocket = new ServerSocket(port:PORT);
            System.out.println("Server started on port " + PORT);

            Arrays.fill(a:board, val:' ');  // Initialize the board

            while (true) {
                if (players.size() < 2) {
                    Socket socket = serverSocket.accept();
                    PlayerHandler player = new PlayerHandler(socket, players.size() + 1);
                    players.add(e:player);
                    new Thread(task:player).start();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Figure 2.5: Initializes game board and waits for client connections

```
private static synchronized boolean checkWin() {
    // Winning combinations
    int[][] winCombos = {
        {0, 1, 2}, {3, 4, 5}, {6, 7, 8},
        {0, 3, 6}, {1, 4, 7}, {2, 5, 8},
        {0, 4, 8}, {2, 4, 6}
    };

    for (int[] combo : winCombos) {
        if (board[combo[0]] != ' ' && board[combo[0]] == board[combo[1]] && board[combo[1]] == board[combo[2]]) {
            return true;
        }
    }
    return false;
}
```

Figure 2.6: Validates moves and checks for the winner

- Client:

```
public TicTacToeClient(String serverAddress) throws Exception {
    socket = new Socket(host: serverAddress, port: 12345);
    out = new PrintWriter(out: socket.getOutputStream(), autoFlush: true);
    in = new BufferedReader(new InputStreamReader(in: socket.getInputStream()));

    for (int i = 0; i < 9; i++) {
        buttons[i] = new JButton(text: "");
        buttons[i].setFont(new Font(name: "Arial", style: Font.PLAIN, size: 60));
        buttons[i].addActionListener(new ButtonClickListener(index: i));
        frame.add(buttons[i]);
    }

    frame.setLayout(new GridLayout(rows: 3, cols: 3));
    frame.setSize(width: 300, height: 300);
    frame.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
    frame.setVisible(b: true);

    new Thread(new Listener()).start();
}

private class ButtonClickListener implements ActionListener {
    private int index;

    public ButtonClickListener(int index) {
        this.index = index;
    }
```

Figure 2.7: Displays the game board and allows players to make moves.

```
private class Listener implements Runnable {
    public void run() {
        try {
            while (true) {
                String response = in.readLine();
                if (response.startsWith(prefix: "WELCOME")) {
                    mySymbol = response.charAt(index: 8);
                } else if (response.startsWith(prefix: "START")) {
                    myTurn = mySymbol == 'X';
                } else if (response.startsWith(prefix: "MOVE")) {
                    int index = Integer.parseInt(response.split(regex: " ")[1]);
                    char symbol = response.split(regex: " ")[2].charAt(index: 0);
                    buttons[index].setText(text: String.valueOf(c: symbol));
                    myTurn = (symbol != mySymbol);
                } else if (response.startsWith(prefix: "WIN")) {
                    JOptionPane.showMessageDialog(parentComponent: frame, "Player " + response.charAt(index: 4) + " wins!")
                    System.exit(status: 0);
                } else if (response.startsWith(prefix: "INVALID")) {
                    JOptionPane.showMessageDialog(parentComponent: frame, message: "Invalid move. Try again.");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public static void main(String[] args) throws Exception {
    String serverAddress = JOptionPane.showInputDialog(message: "Enter Server Address:");
    new TicTacToeClient(serverAddress);
}
}
```

Figure 2.8: Communicates the server to update the board

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The Weather Application and Tic-Tac-Toe game were created using Java and tested in a real-time environment. The setup allowed evaluation of socket communication, GUI responsiveness, and overall functionality.

3.1.1 Setup and Testing Procedure

The project was implemented using Java programming language with two primary components: the Weather Application and the Tic-Tac-Toe game (client-server). The following steps were undertaken to set up and test the application:

- **Weather Application:**

(a) **Setup:** Developed using Swing for GUI and connected to the OpenWeatherMap API for weather data retrieval. A valid API key was used.

(b) **Testing:** The application was tested by entering different city names to verify the correct weather and temperature were displayed.

- **Tic-Tac-Toe Game:**

(a) **Setup:** The server was implemented using Java sockets, capable of handling two simultaneous client connections. The client side used Swing for the graphical representation of the game board.

(b) **Testing:** The game was tested by connecting two clients to the server, simulating moves, and verifying the synchronization of game states across clients.

Key Steps of Setup:

- **Installing Required Tools:**

(a) Installed Java JDK and NetBeans IDE for code development and testing.

(b) Configured OpenWeatherMap API key for Weather Application.

- **Code Implementation:**

(a) Weather Application: Implemented using Swing for GUI and HTTPURL-Connection for API interaction.

(b) Tic-Tac-Toe Game: Implemented socket-based communication for the server and client, along with GUI for player interaction.

- **Testing Environment:**

(a) Verified weather data retrieval by entering valid and invalid city names.

(b) Tested Tic-Tac-Toe moves, game result validation, and error handling for invalid inputs.

3.2 Results Analysis/Testing

3.2.1 Weather Application

- **Interface:** The first interface prompted the user to input a city name and displayed the weather conditions and temperature after fetching data from the API.

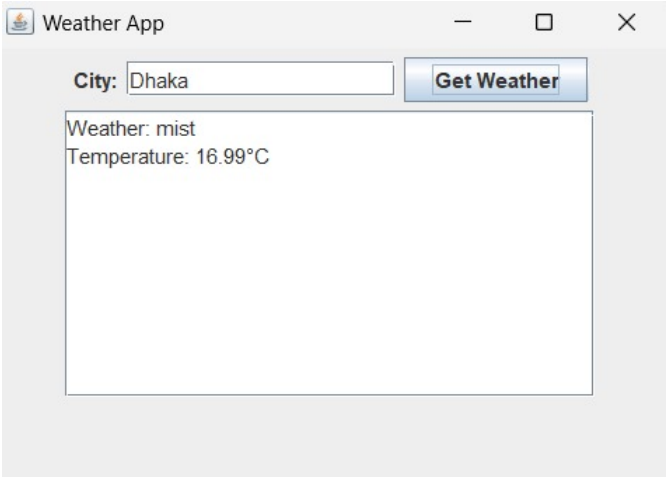

Figure 3.1: Weather Application Interface

3.2.2 Tic-Tac-Toe Game

- **First Interface:**The game welcomed each player with their assigned symbol ('X' or 'O') and started the game upon the second player's connection.

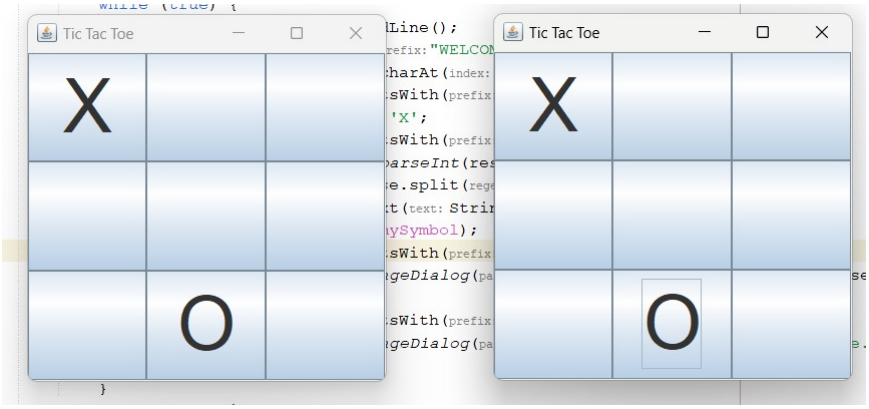

Figure 3.2: First Interface

Game Execution:

- Players made moves alternately. The board updated in real time for both players. When a user works, he can't choose two options at a time. Two users will work one at a time and will continue to update.

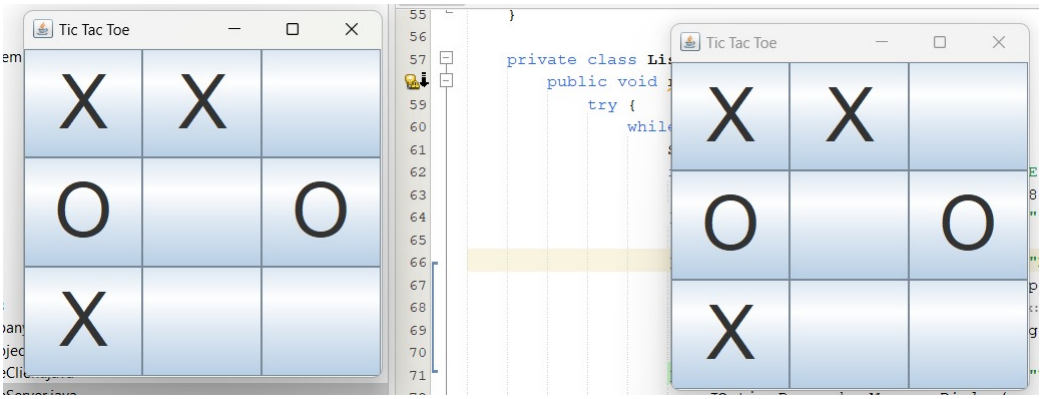

Figure 3.3: The board updated in real time

- The game displayed a message declared the winner if a player formed a winning combination.

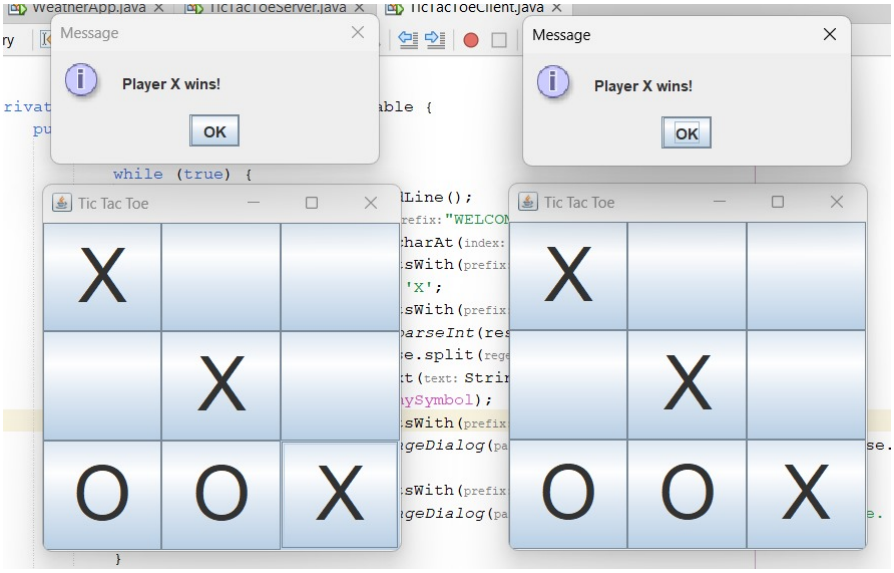

Figure 3.4: A message declared the winner

3.3 Results Overall Discussion

3.3.1 How It Worked

- **Weather Application:** Successfully fetched weather data for user-provided cities and displayed it in a structured format. The GUI was responsive and user-friendly.
- **Tic-Tac-Toe Game:** Enabled two players to play the game over a network, synchronized moves, and determined the winner or draw state effectively.

3.3.2 Problems Find

Weather Application: Handling API connection errors when the city name was invalid or internet connectivity was lost.

Tic-Tac-Toe Game: Synchronization issues when clients disconnected unexpectedly. Ensuring that invalid moves were correctly identified and handled.

3.3.3 Complex Engineering Problem Discussion

Weather Application:

- Parsing JSON responses from the API and displaying meaningful messages for users.

Tic-Tac-Toe Game:

- Managing client-server communication for real-time game updates.
- Ensuring thread-safe operations when multiple clients accessed shared resources like the game board.

Chapter 4

Conclusion

4.1 Discussion

This project demonstrated the successful implementation of a Weather Application using an API and a networked Tic-Tac-Toe game using sockets. The Weather Application provided a simple and user-friendly interface to access real-time weather information. The Tic-Tac-Toe game offered a multiplayer experience with synchronized gameplay and dynamic updates.

4.2 Limitations

Weather Application:

- Relied on API availability and internet connectivity.
- Could not handle advanced weather queries

Tic-Tac-Toe Game:

- Required manual server restart for a new game session.
- Limited to two players.

4.3 Scope of Future Work

Weather Application:

- Add graphical representation of weather data.
- Enhance error handling for invalid inputs and connection issues.

Tic-Tac-Toe Game:

- Support for reconnecting clients during an ongoing game.
- Extend to support more players or implement an AI opponent.

References

[1] Maisha Muntaha. Lecture on computer networking lab, 2024. Class Lecture.

[2] Api learning for project:. https://medium.com/@archimedes.fagundes/5-ways-to-call-an-api-in-java-b3de65fb2022.

[3] Socket learning:. https://www.baeldung.com/a-guide-to-java-sockets.

[4] Swing in java. https://docs.oracle.com/javase/tutorial/uiswing/.