VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NET-Centric
IT076IU

PROJECT REPORT

Topic: Pokemon Game

By Group 40 – Member List

No.	Full name – Student name	Student ID
1	Nguyễn Hoàng Minh Khôi	ITITIU21229
2	Nguyễn Trần Gia Huy	ITITIU21054

Instructor: Assoc. Prof. Nguyen Van Son

INTRODUCTION

This report illustrates the main contents of a Pokemon game built by Golang Programming Language. This project addresses different aspects of the possibilities of the Golang Programming Language throughout the Net-Centric course. The Pokemon game covers the function of scraping pokemon data from the website, battling and catching games.

The primary goal of this project is to forward the capabilities and versatilities of the Golang Programming Language in portraying a Pokemon game that can handling various tasks, scraping, concurrent programming and also works real-time multiplayerly.

POKEDEX

I. Overview

The Golang Pokedex is a command-line tool that scrapes Pokémon data from **Pokedex.org** using the Playwright-Go library. It automates the scraping process, retrieves detailed Pokémon information, and exports it to a JSON file for easy access and analysis. This project demonstrates the use of Golang for web automation, data extraction, and file handling.

II. Structure

Data Structures

The Go file defines some Go structs to clarify the data of each Pokemon:

- **Stats:** Includes HP, Attack, Defense, Sp Attack, Sp Defend, Speed.
- **GenderRatio:** Presenting the male and female ratios.
- **Profile:** Includes Height. Weight, CatchRate, etc.
- **DamageWhenAttacked:** Pokemon Type and Efficient Typing.
- **Moves:** Includes Moves that a Pokemon can learn.
- **Pokemon:** Main struct combining the above information.

Constants

- numberOfPokemons: The number of Pokemon in total needed scrape.
- baseURL: URL of the website to scrape.

Variables

- pokemons: store pokemon data after scraping.

III. Implementation

Main Function

The main function initiates the scraping process by calling the crawlPokemonsDriver function with the number of Pokémon to scrape.

crawlPokemonsDriver Function

1. Initializes Playwright and launches a Chromium browser instance.
2. Navigates to the base URL.
3. Iterates through the specified number of Pokémon.
4. For each Pokémon, simulates a button click to open its details and calls crawlPokemons to scrape the data.
5. After scraping, converts the collected data to JSON and writes it to a file.
6. Closes the browser and stops Playwright.

crawlPokemons Function

This function extracts detailed information for a single Pokémon:

- **Stats:** Extracts statistics like HP, Attack, Defense, etc.
- **Profile:** Extracts profile details including height, weight, catch rate, etc.
- **GenderRatio**: Extracts the male and female ratio.
- **DamageWhenAttacked:** Extracts the elements and their damage coefficients.
- **Evolution**: Extracts evolution details.
- **Moves:** Extracts details about the moves a Pokémon can learn.
- **Elements:** Extracts the Pokémon's elements (types).

Finally, the scraped data for the Pokémon is appended to the pokemons slice.

IV. USAGE

1. Install the Playwright-Go library.
2. Move to crawler directory with cd crawler
3. Run the script using go run crawler.go
4. The script will navigate through the pokedex.org and then crawl all of the data.

V. Challenges and Solutions

Challenge: Suitable Library

● Solution: After many failed attempts with Seleniumand and rollycolly libraries cause they only scrape the first URL page, we decided to use Playwright as the most efficient way to scrap.

PokeBat

I. Introduction

The PokeBat project is a TCP-based multiplayer Pokémon battle game where players can authenticate, receive random Pokémon, and engage in battles. The server handles player authentication, assigns Pokémon, and facilitates battles. The client connects to the server, allows user interaction, and processes game instructions.

II. Objectives

Implement a TCP server to handle multiple players and facilitate Pokémon battles.
Allow players to authenticate and join the game.
Assign random Pokémon to players and manage battles.
Ensure smooth communication between the server and clients.

III. Project Structure

Directories

- **server/:** Contains the server-side code for handling game logic.
- **client/:** Contains the client-side code for player interactions.
- **assests/:** Contains JSON files for user credentials and Pokémon data.

Data Structures

The project defines several Go structs to model the game data:

- **User **: Represents a player with a username and password.
- **Stats:** Holds Pokémon statistics like HP, Attack, Defense, etc.
- **Profile:** Contains attributes like height, weight, catch rate, etc.
- **Damage:** Stores elements and their damage coefficients.

- **Pokemon:** The main struct that aggregates all the above information.
- **Player:** Represents a player with attributes like name, connection, Pokémon, and active Pokémon index.
- **Battle:** Represents a battle between two players.

Constants

- HOST: The host address for the server.
- PORT: The port on which the server listens.
- TYPE: The type of connection (TCP).
- POKEDEX_FILE: Path to the file containing Pokémon data.
- USER_FILE: Path to the file containing user data.

IV. Implementation

Server (server/server.go)

- **Main Function**
    1. Starts the TCP server and listens for incoming connections.
    2. Authenticates players and adds them to the player list.
    3. Loads Pokémon data from a file.
    4. Assigns random Pokémon to players once the minimum player requirement is met.
    5. Initiates a battle between the first two players.

- **Helper Functions**
    1. authenticate: Verifies user credentials.
    2. loadUsers: Loads user data from a JSON file.
    3. loadPokemons: Loads Pokémon data from a JSON file.
    4. chooseStartingPokemon: Prompts the player to choose their starting Pokémon.
    5. pokemonBattle: Manages the battle logic, including player turns and actions.
    6. performAttack: Handles the attack logic during a battle.
    7. switchPokemon: Allows a player to switch their active Pokémon.
    8. endBattle: Ends the battle and notifies the players of the result.

Client (client/client.go)

1. Connects to the server using TCP.
2. Prompts the user for authentication details.
3. Sends authentication data to the server and processes the response.
4. Listens for game instructions from the server and sends user inputs.
5. Displays game messages and handles game flow based on server instructions.

V. Usage

Running the Server

1. Ensure Go is installed on your system.
2. Navigate to the server directory.
3. Run the server using go run server.go.

Running the Client

1. Navigate to the client directory.
2. Run the client using go run client.go.
3. Enter your username and password to authenticate and join the game.

VI. Challenges and Solutions

Challenge: Handling Multiple Players

- Solution: Used Go's net package to handle multiple TCP connections and manage player interactions concurrently.

Challenge: Authenticating Users

- Solution: Implemented a simple authentication mechanism using username and password, with password hashing for security.

Challenge: Managing Game State

- Solution: Maintained separate structs for players and battles to keep track of the game state and ensure smooth transitions between actions.

Challenge: Ensuring Real-time Communication

- Solution: Used blocking reads and writes on TCP connections to facilitate real-time communication between the server and clients.

POKECAT

I. Introduction

The PokeCat project is a multiplayer Pokémon game that utilizes HTTP for communication between clients and the server. Players can register, log in, and interact with the game world, including moving around, saving their progress, and viewing the game grid. The server manages player sessions, game state, and Pokémon data.

II. Objectives

Implement an HTTP server to handle player requests and manage game state.
Allow players to register and log in to their accounts.
Enable players to move within the game world and interact with Pokémon.
Provide a grid view of the game world to players.
Ensure secure handling of user credentials and game data.

III. Project Structure

Directories

- **server/:** Contains the server-side code for handling game logic and player interactions.
- **client/:** Contains the client-side code for user interactions and game commands.
- **playerData/:** Contains JSON files for user credentials and Pokémon data.

Data Structures

The project defines several Go structs to model the game data:

- **User **: Represents a registered player with a username, password hash, and player ID.
- **Stats:** Holds Pokémon statistics like HP, Attack, Defense, etc.
- **Profile:** Contains attributes like height, weight, catch rate, etc.
- **DamageCoefficient:** Stores elements and their damage coefficients.
- **Pokemon:** The main struct that aggregates all the above information.
- **Player:** Represents a player with attributes like name, position, caught Pokémon, and auto mode status.
- **GameState:** Represents the overall state of the game, including players and Pokémon on the grid.

Constants

- Host: The host address for the server.
- Port: The port on which the server listens.

IV. Implementation

Server (server/server.go)

- **Main Function**
    1. Initializes the game state and loads Pokémon data.
    2. Starts the HTTP server and listens for incoming requests.
    3. Handles player registration, login, and joining the game.
    4. Manages player movements and interactions with Pokémon.
    5. Provides a grid view of the game world for players.

- **Handler Functions**
    1. handlePlayerRegister: Registers a new player and saves their data.
    2. handlePlayerLogin: Authenticates a player and returns their player ID.
    3. handlePlayerJoin: Adds a player to the game state.
    4. handlePlayerMove: Updates the player's position based on their movement command.
    5. handleDebugGrid: Displays the grid around the player, showing Pokémon and other players.
    6. handlePlayerSave: Saves the player's current game state.

Client (client/client.go)

- Connects to the server using HTTP.
- Prompts the user for registration or login details.
- Sends requests to the server for authentication and game actions.
- Displays game messages and handles user inputs for movement and commands.
- Provides a user-friendly interface for interacting with the game.

V. Usage

Running the Server

1. Ensure Go is installed on your system.
2. Navigate to the server directory.
3. Run the server using go run server.go.

Running the Client

1. Navigate to the client directory.
2. Run the client using go run client.go.
3. Follow the prompts to register or log in and start playing.

VI. Challenges and Solutions

Challenge: Handling Multiple Players

- Solution: Used Go's HTTP package to manage multiple player sessions and requests concurrently.

Challenge: Authenticating Users

- Solution: Implemented a secure authentication mechanism using hashed passwords and user data validation.

Challenge: Managing Game State

- Solution: Utilized structs to maintain separate data for players and Pokémon, ensuring smooth transitions and interactions.

Challenge: Providing Real-time Updates

- Solution: Implemented a grid view that updates based on player movements and interactions, allowing for dynamic gameplay.

CONCLUSION

In conclusion, this project successfully integrated the Pokedex, PokeCat, and PokeBat while enhancing our understanding of Golang. We utilized web scraping techniques with libraries such as Playwright to efficiently gather data for the Pokedex. The project involved creating various functions to manage game mechanics and user interactions, demonstrating the use of struct types for organizing data and implementing methods for functionality. We also learned about handling JSON data for storing and retrieving Pokémon attributes, as well as managing concurrent processes with Goroutines. Overall, this project provided us with practical experience in coding techniques and best practices in Golang, reinforcing our programming skills and understanding of the language.