Siedlce University of Natural Sciences and Humanities

Project assignment from laboratory exercises in the subject:

Technologie sieciowe

Simple Service Mesh Management Protocol (SSMMP)

Major: Computer Science

Jakub Żurawicki

Jakub Bonda

Mateusz Kielak

Siedlce 2023

Topic of the analysis

The subject of the paper is to create the Simple Service Mesh Management Protocol (SSMMP) as a specification for deployment in a Cloud cluster, with the main goal of providing a platform supporting the required functionalities. The specification consists of message formats exchanged between protocol conversion parties (actors) and actions taken by message senders and receivers. The actors include: Manager, agents (residing on nodes forming the cluster), and instances of microservices operating on these nodes. The project must include elements such as registration and login handling, account management in the database, displaying the latest 10 chat messages in a table format, a chat enabling message sending and saving them in the database, and an HD file server allowing file upload and download. The project implementation will be carried out in Java, utilizing multithreading. The database will be created in MySQL.

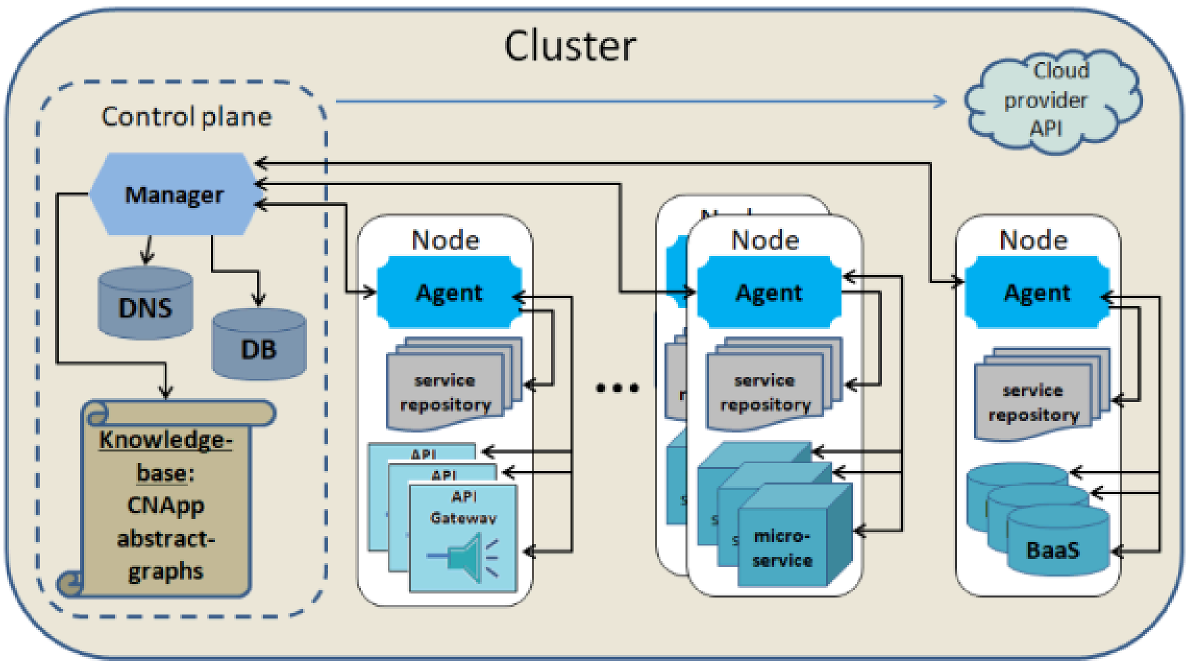
Execution:

Jakub Żurawicki

Jakub Bonda

Mateusz Kielak

Functional Architecture



Basic project schema

User Interface - Command Line Interface (CLI)

CLI - Command-Line Interface (CLI). It has the following functionalities (modes):

- **Registering a new user**
- **User login**
- **Adding posts (chat)**
- **Displaying the post board (chat): 10 latest messages**
- **File Transfer::**
    - Sending files from a user-specified path to a special disk server for general filing.
    - Downloading a file with a user-specified name.
- **HELP:** Basic information about the project and its usage.

MicroserviceManager

The `MicroserviceManager` class is responsible for managing microservices within the application. It handles the registration, deletion, and port allocation for microservices, providing a centralized management system.

The Manager communicates only with agents. An agent in a node communicates with all service instances operating in that node. Within SSMMP, each service instance (operating in a node) can only communicate with its agent in that node. All executions of service instances and the termination of running instances are controlled by the Manager through its agents. Each agent must register with the Manager so that the network address of its node and the service repository are known to the Manager.

The class follows the Singleton pattern, ensuring that only one instance of `MicroserviceManager` exists throughout the application.

Fields:

- `microservicePorts`: A map that associates microservice names with their respective ports.

- `portApi`: The default port for the API, set to 3000.

Public Methods:

1. `getInstance()`: Retrieves the singleton instance of `MicroserviceManager`.

2. `addMicroservice(String serviceName)`: Adds a microservice with the specified service name.

3. `deleteMicroservice(String serviceName)`: Deletes a microservice identified by its service name.

4. `getMicroservicePort(String serviceName)`: Retrieves the port number associated with a microservice.

5. `getApiPort()`: Retrieves the port number for the API.

Private Methods:

1. `registerMicroservice(String serviceName)`: Registers a microservice by assigning it a unique port and starting it.

2. `unregisterMicroservice(String serviceName)`: Unregisters a microservice by removing it from the port map.

3. `findAvailablePort()`: Finds an available port for a microservice.

4. `isPortOccupied(int port)`: Checks if a port is occupied by attempting to create a `ServerSocket` on that port.

This class serves as a crucial component in managing the lifecycle and communication of microservices within the application's architecture.

AGENT

The Agent class is responsible for managing microservice threads within the application. It facilitates the starting, stopping, and retrieving of ports for microservices. *Agent* - In a node, it communicates with all service instances operating in that node. On the Manager's request, the agent can start service instances whose bytecode is available in its repository or terminate those instances.

Fields:

- `microserviceThreads`: A map that stores microservice threads, indexed by microservice names.

- `microserviceManager`: An instance of the `MicroserviceManager` class responsible for managing microservice ports.

Public Methods:

1. `getInstance()`: Retrieves the single instance of `Agent`.

2. `startMicroservice(String serviceName)`: Starts a microservice with the given service name.

3. `stopMicroservice(String serviceName)`: Stops a microservice with the given service name.

4. `getPort(String serviceName)`: Retrieves the port of the microservice with the given service name using the `MicroserviceManager`.

The `Agent` class acts as an intermediary between the microservice management module and the microservices themselves, providing control over their lifecycle.

AgentClient

The AgentClient class serves as a client for accessing information from the MicroserviceManager, particularly the API port. The AgentClient class provides a simplified interface for accessing the API port information from the MicroserviceManager.

Public Methods:

1. `getInstance()`: Retrieves the single instance of AgentKlient.

2. `getPort()`: Retrieves the API port using the MicroserviceManager.

API Gateway

API Gateway - CNApp entry points for users. Typically, an API Gateway has only one component. Its functionality involves forwarding user requests to the appropriate microservices. The API Gateway is stateless, operating without maintaining state (transferring byte streams between various components, both from microservices to clients and from clients to microservices). It maintains connections with clients using threads.

Public Methods:

1. Method `run()`: -The `run()` method is responsible for handling the client connection via the server socket. Upon accepting a connection, this method creates a `BufferedReader` object for reading input data from the client and a `PrintWriter` object for sending output data to the client. It then reads lines of requests from the client using the `readLine()` method and passes them to the `forwardRequest()` method for processing. Responses returned by `forwardRequest()` are sent back to the client using the `PrintWriter` object. In case of input-output errors (IOException), the method catches the exception and prints it to the standard error output.

Private Methods:

1. Method `forwardRequest(String request, String serviceName, PrintWriter out)`: The `forwardRequest()` method is responsible for forwarding requests from the client to the appropriate microservices based on the request type. Upon receiving a request from the client, this method registers and starts the corresponding microservice using the `MicroserviceManager` class. It then retrieves the port on which this microservice is running and creates a client socket for communicating with the microservice. The request is forwarded

to the microservice via the client socket, and the response is received and forwarded back to the client using the `PrintWriter` object. Finally, the microservice is stopped and unregistered using the `MicroserviceManager`. In case of input-output errors (IOException) or thread interruption (InterruptedException), the method catches the exception and prints it to the standard error output.

Microservices in the application are also stateless, meaning they do not store state between requests and pass data (bytes) either to other microservices or to end users.

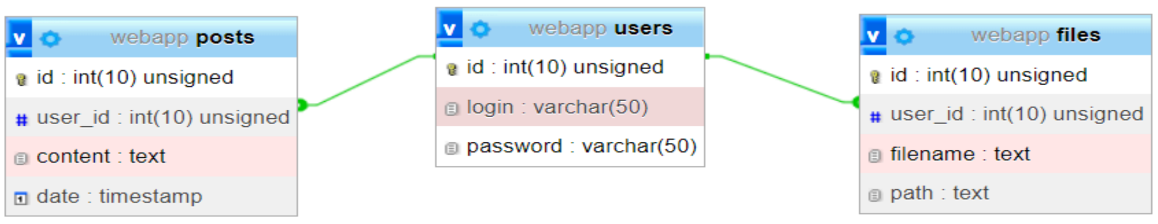
Types of microservices:

1. User login
2. User registration
3. Chat - available only for logged-in users, allows adding new entries to the board
4. Board - accessible without the need for logging in, displays the latest 10 posts
5. File transfer - available only for logged-in users, allows sending and downloading specific files.

Database Project

The database is written in MySQL and is named "ProjektTS". It is used to store information about users, files in the system, and archive user posts. It consists of three tables: "files", "users", and "chat". The detailed structure of this database is described below.

Graphical Representation of the Database



Database code

```sql
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Baza danych: `webapp`
--

-- --------------------------------------------------------

--
-- Struktura tabeli dla tabeli `files`
--

CREATE TABLE `files` (
  `id` int(10) UNSIGNED NOT NULL,
  `user_id` int(10) UNSIGNED NOT NULL,
  `filename` text NOT NULL,
  `path` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Zrzut danych tabeli `files`
--

INSERT INTO `files` (`id`, `user_id`, `filename`, `path`) VALUES
(2, 1, 'mysql-connector-j-8.2.0.jar', 'pliki\\login\\mysql-connector-j-8.2.0.jar'),
(4, 3, 'Czystarlinkimogastacsiealternatywadlagps.pptx', 'pliki\\loginn\\Czystarlinkimogastacsiealternatywadlagps.pptx'),
(5, 3, '380210059_284015844511318_3839805751918941074_n.png', 'pliki\\loginn\\380210059_284015844511318_3839805751918941074_n.png'),
(9, 2, 'l2z3.html', 'pliki\\login1\\l2z3.html'),
(10, 2, 'API.png', 'pliki\\login1\\API.png');

-- --------------------------------------------------------

--
-- Struktura tabeli dla tabeli `posts`
--

CREATE TABLE `posts` (
  `id` int(10) UNSIGNED NOT NULL,
  `user_id` int(10) UNSIGNED NOT NULL,
  `content` text NOT NULL,
  `date` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Zrzut danych tabeli `posts`
--

INSERT INTO `posts` (`id`, `user_id`, `content`, `date`) VALUES
(1, 1, 'testowy post od logina', '2023-11-19 11:09:51'),
(2, 2, 'pozdro od login1', '2023-11-19 11:12:56'),
(3, 2, 'testtesttest', '2023-11-19 11:38:57'),
(4, 1, 'post testowy', '2023-11-19 17:15:49'),
(5, 1, 'post z rana', '2023-11-22 04:33:24'),
(6, 3, 'test o 6:10', '2023-11-22 05:10:31');

-- --------------------------------------------------------

--
-- Struktura tabeli dla tabeli `users`
--

CREATE TABLE `users` (
  `id` int(10) UNSIGNED NOT NULL,
  `login` varchar(50) CHARACTER SET utf8 COLLATE utf8_polish_ci NOT NULL,
  `password` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
```

```
-- Zrzut danych tabeli `users`
--

INSERT INTO `users` (`id`, `login`, `password`) VALUES
(1, 'login', 'hasslo'),
(2, 'login1', 'hasslo'),
(3, 'loginn', 'hasslo');

--
-- Indeksy dla zrzutów tabel
--

--
-- Indeksy dla tabeli `files`
--
ALTER TABLE `files`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indeksy dla tabeli `posts`
--
ALTER TABLE `posts`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`),
  ADD KEY `user_id_2` (`user_id`);

--
-- Indeksy dla tabeli `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT dla zrzuconych tabel
--

--
-- AUTO_INCREMENT dla tabeli `files`
--
ALTER TABLE `files`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;
```

```

--
-- AUTO_INCREMENT dla tabeli `posts`
--
ALTER TABLE `posts`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;

--
-- AUTO_INCREMENT dla tabeli `users`
--
ALTER TABLE `users`
  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;

--
-- Ograniczenia dla zrzutów tabel
--

--
-- Ograniczenia dla tabeli `files`
--
ALTER TABLE `files`
  ADD CONSTRAINT `user_idd` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Ograniczenia dla tabeli `posts`
--
ALTER TABLE `posts`
  ADD CONSTRAINT `user_iddd` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

General Operation Scheme:

<u>User Login:</u>

1. The user enters their login credentials, i.e., username and password.

2. A request is sent to the login microservice through the API, and the data is appropriately interpreted.

3. The server authorizes this data by querying the "users" table in the database.

4. If the data is correct, the server grants access to resources only for logged-in users. The response contains the operation status, which, in turn, determines the user's login status stored in the "log" variable.

<u>User Registration:</u>

1. A new user enters their data, including username and password.

2. A request is sent to the registration microservice through the API, and the data is appropriately interpreted.

3. The server checks whether the login is not already in use by another user.

4. If the data is unique, the server adds the new user to the "users" table in the database.

5. A response is returned to the user with the appropriate status.

<u>Chat:</u>

1. The chat is available only for logged-in users. The "log" variable is checked.

2. If the previous point is true, the user can enter a message.

3. A request is sent to the login microservice through the API, and the data is appropriately interpreted.

4. Chat messages are stored in the "chat" table in the database, including information about the author, content, and time. The server sends a command to the database.

5. The status of whether the operation was successful is returned.

Board:

1. The board is available without the need for logging in.

2. A request is sent to the login microservice through the API, and the data is appropriately interpreted.

3. The server retrieves the latest 10 posts from the "chat" table that can be read.

4. In the response, the server provides the 10 latest messages to the user, and the data is interpreted and displayed in the console if the operation status is successful.

File Transfer:

1. Available only for logged-in users. The "log" variable is checked.

2. If the previous point is true, the user can perform the appropriate operation or send a request with the file name for download or the given file path for file upload, depending on the selected option.

3. A request is sent to the file transfer microservice through the API, and the data is appropriately interpreted.

4. The server connects to the database and retrieves values from the "files" table.

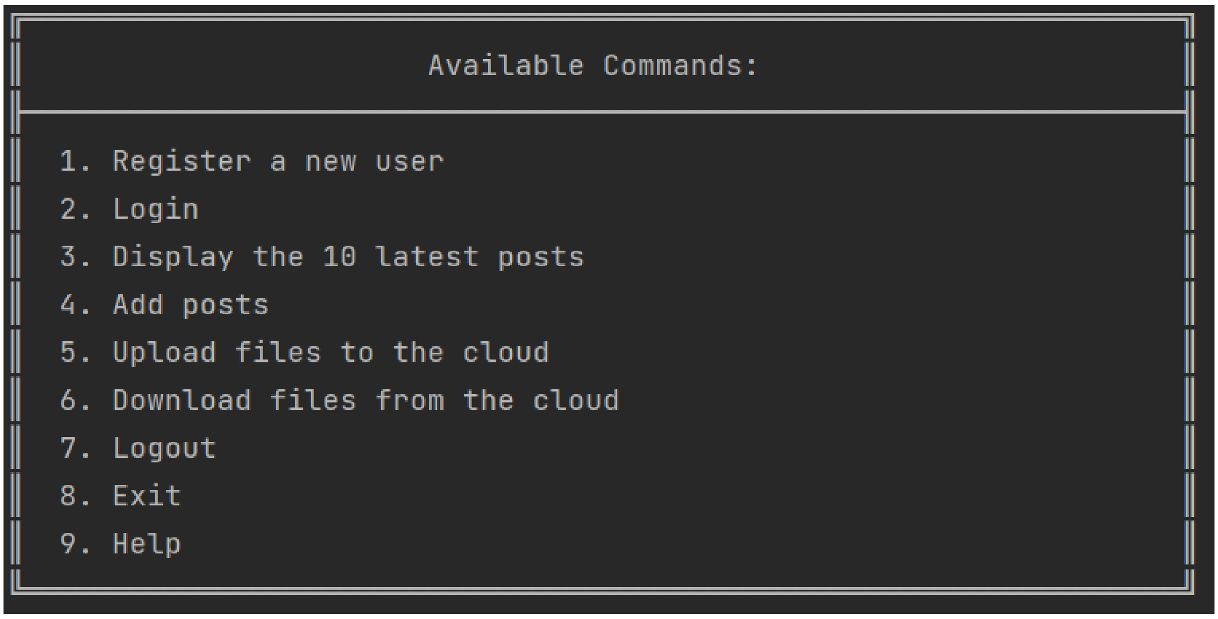
5. Then, from the server based on the saved URL address, the appropriate file is sent to the user by sending a response according to the protocols.

6. On the user's side, the file is decoded and the saveFile() function is called to save the file.

7. In the case of sending a file based on the given path, the file is encoded in Base64, and then a request is sent according to the protocols to the file transfer microservice where we interpret the request through the API and then send

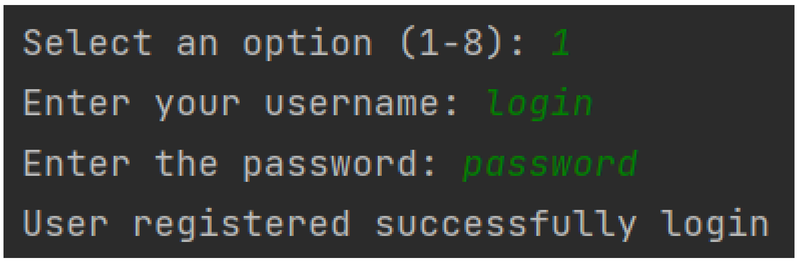
packets to the file server to save it in the folder. Data about the file and its URL address are stored in the database in the "files" table.

8. Finally, a message with the appropriate status is returned to the client.



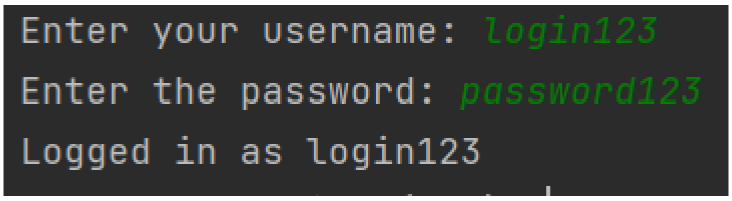
```
Available Commands:

1. Register a new user
2. Login
3. Display the 10 latest posts
4. Add posts
5. Upload files to the cloud
6. Download files from the cloud
7. Logout
8. Exit
9. Help
```

Screenshot number 1 depicts the client-side application menu in English.



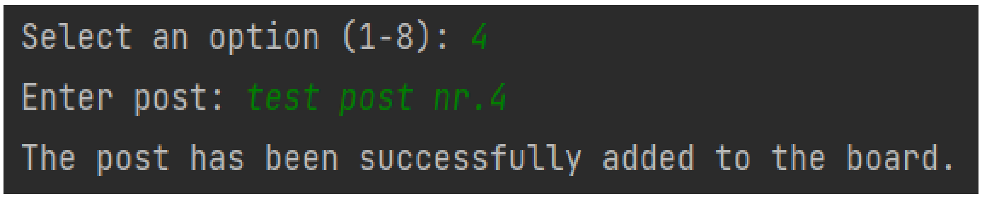
```
Select an option (1-8): 1
Enter your username: login
Enter the password: password
User registered successfully login
```

Screenshot number 2 depicts the login from the client-side application menu.



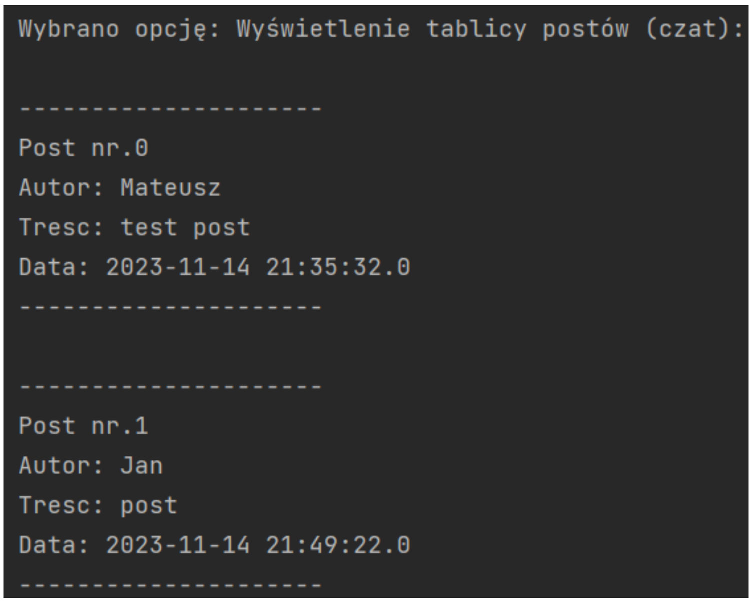
```
Enter your username: login123
Enter the password: password123
Logged in as login123
```

Screenshot number 3 depicts the registration from the client-side application menu.



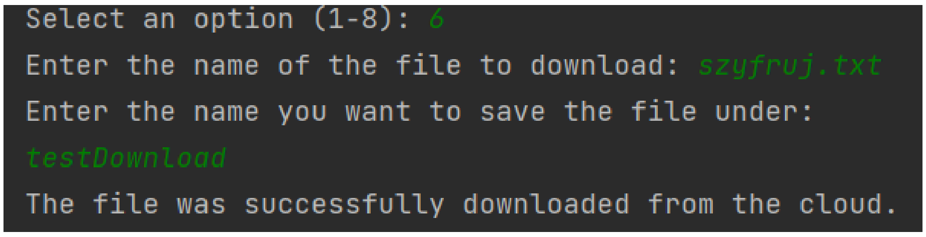
```
Select an option (1-8): 4
Enter post: test post nr.4
The post has been successfully added to the board.
```

Screenshot number 4 depicts adding a post from the client-side application menu.



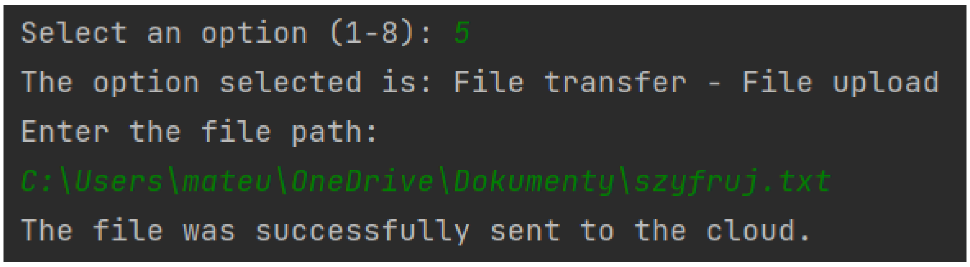
```
Wybrano opcję: Wyświetlenie tablicy postów (czat):

--------------------
Post nr.0
Autor: Mateusz
Tresc: test post
Data: 2023-11-14 21:35:32.0
--------------------


--------------------
Post nr.1
Autor: Jan
Tresc: post
Data: 2023-11-14 21:49:22.0
--------------------
```

Screenshot number 5 depicts the posts table from the client-side application menu.



```
Select an option (1-8): 6
Enter the name of the file to download: szyfruj.txt
Enter the name you want to save the file under:
testDownload
The file was successfully downloaded from the cloud.
```

Screenshot number 6 depicts downloading files from the client-side application menu.



```
Select an option (1-8): 5
The option selected is: File transfer - File upload
Enter the file path:
C:\Users\mateu\OneDrive\Dokumenty\szyfruj.txt
The file was successfully sent to the cloud.
```

Screenshot number 7 depicts sending files from the client-side application menu.

Sources of information:

- Materials provided by Prof. Dr. Hab. Stanisław Michał Ambroszkiewicz
- Java. Basics. 12th Edition - Cay Horstmann

THE END

Computer Science UPH

3rd semester, year 23/24