

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-5-project-milestone-1/grade/rev>

Course: IT114-003-F2024

Assignment: [IT114] Module 5 Project Milestone 1

Student: Ricardo V. (rev)

Submissions:

Submission Selection

1 Submission [submitted] 10/16/2024 10:36:03 PM ▾

Instructions

[^ COLLAPSE ^](#)

Overview Video: <https://youtu.be/A2yDMS9TS1o>

1. Create a new branch called Milestone1
2. At the root of your repository create a folder called Project if one doesn't exist yet
 1. You will be updating this folder with new code as you do milestones
 2. You won't be creating separate folders for milestones; milestones are just branches
3. Copy in the code from Sockets Part 5 into the Project folder (just the files)
 2. <https://github.com/MattToegel/IT114/tree/M24-Sockets-Part5>
4. Fix the package references at the top of each file (these are the only edits you should do at this point)
5. Git add/commit the baseline and push it to github
6. Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
7. Ensure the sample is working and fill in the below deliverables 1. Note: Don't forget the client commands are /name and /connect
8. Generate the output file once done and add it to your local repository
9. Git add/commit/push all changes
10. Complete the pull request merge from the step in the beginning
11. Locally checkout main
12. git pull origin main

Branch name: Milestone1

Group

Group: Start Up

Tasks: 2

Points: 3

100%

▲ COLLAPSE ▾

Task

Group: Start Up

Task #1: Start Up

Weight: ~50%

Points: ~1.50

100%

▲ COLLAPSE ▾

i Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.



Columns: 1

Sub-Task

Group: Start Up

Task #1: Start Up

Sub Task #1: Show the Server starting via command line and listening for connections

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1

A screenshot of a terminal window titled 'Terminal' showing command-line output. The logs indicate the server is starting up and listening for connections. Key text visible includes 'Starting in port 8080', 'Listening in port 8080', and 'Server listening in port 8080'.

Server starting via command line and listening for connections

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Start Up

Task #1: Start Up

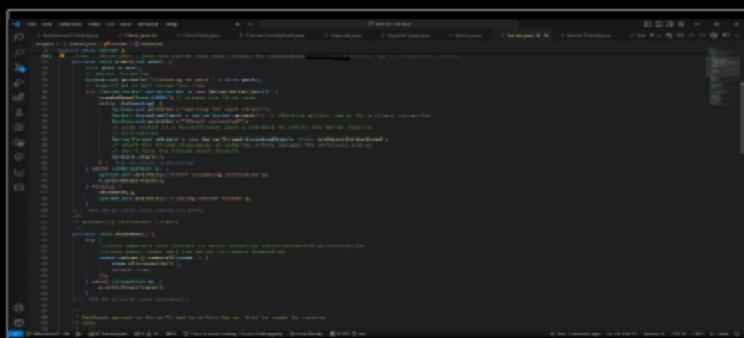
Sub Task #2: Show the Server Code that listens for connections

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1



A screenshot of a Java code editor showing a file named `Server.java`. The code contains a main method that starts a server on port 8080 and handles incoming connections by printing a message to the console. The code uses `System.out.println` to output "Hello from Client Thread".

the Server Code that listens for connections

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code related to starting up and waiting for connections

Response:

The code sets up a basic server that listens for connections on the local host and handles each connection by printing a message.

Sub-Task

Group: Start Up

100%

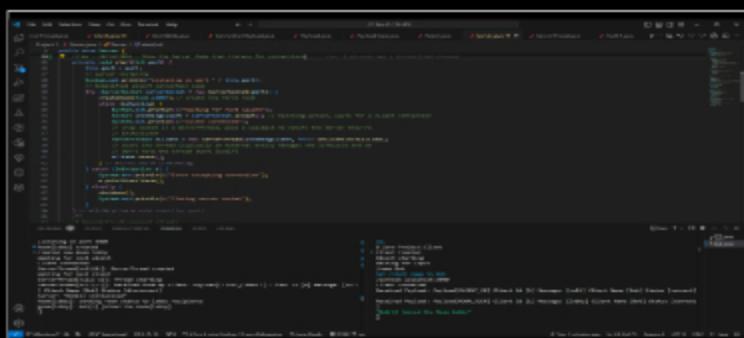
Task #1: Start Up

Sub Task #3: Show the Client starting via command line

Task Screenshots

Gallery Style: 2 Columns

4 2 1



A screenshot of a Java code editor showing a file named `Client.java`. The code contains a main method that creates a socket connection to the server at localhost port 8080. It then reads a message from the server and prints it to the console. The message "Hello from Client Thread" is visible in the output window.

the Client starting via command line

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Start Up

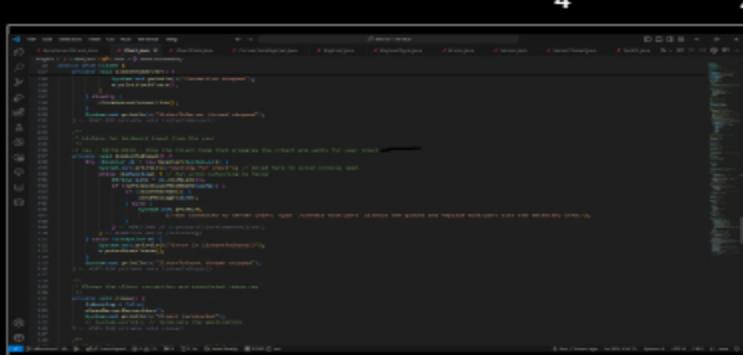
100%

Task #1: Start Up

Sub Task #4: Show the Client Code that prepares the client and waits for user input

Task Screenshots

Gallery Style: 2 Columns



A screenshot of a Java code editor showing a class named Client. The code includes imports for java.net, java.util, and javax.swing. It defines a main method that creates a new Client object and calls its start() method. The start() method initializes a socket connection to a host at port 12345 and prints the response to the console.

```
import java.net.*;
import java.util.*;
import javax.swing.*;

public class Client {
    public static void main(String[] args) {
        Client client = new Client();
        client.start();
    }
}

class Client {
    public void start() {
        try {
            Socket socket = new Socket("localhost", 12345);
            System.out.println(socket.getInputStream().readLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

the Client Code that prepares the client and waits for user input

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow leading up to and including waiting for user input

Response:

The code here prompts the user to enter the hostname and port number.

End of Task 1

Task

Group: Start Up



Task #2: Connecting

Weight: ~50%

Points: ~1.50

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

Group: Start Up

Task #2: Connecting

Sub Task #1: Show 3 Clients connecting to the Server

Task Screenshots

Gallery Style: 2 Columns

4

2

1

3 Clients connecting to the Server

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up

Task #2: Connecting

Sub Task #2: Show the code related to Clients connecting to the Server (including the two needed commands)

Task Screenshots

Gallery Style: 2 Columns

4

2

1

the code related to Clients connecting to the Server
(including the two needed commands)

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow

Response:

The client creates a Socket object to connect to the server using the server's IP address and port number and then uses input and output streams to communicate with the server.

End of Task 2

End of Group: Start Up

Task Status: 2/2

Group

Group: Communication

Tasks: 2

Points: 3

100%

▲ COLLAPSE ▲

Task

Group: Communication

Task #1: Communication

Weight: ~50%

Points: ~1.50

100%

▲ COLLAPSE ▲

ⓘ Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

Group: Communication

Task #1: Communication

Sub Task #1: Show each Client sending and receiving messages



Task Screenshots

Gallery Style: 2 Columns

4 2 1



each Client sending and receiving messages

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Communication

Task #1: Communication

Sub Task #2: Show the code related to the Client-side of getting a user message and sending it over the socket



Task Screenshots

4

2

1

The screenshot shows a Java IDE interface with two panes. The left pane displays a file tree with various Java files and resources. The right pane shows a code editor with Java code. A specific line of code is highlighted with a red rectangle, and a black arrow points from the text "the code related to the Client-side of getting a user message and sending it over the socket" to this highlighted line.

the code related to the Client-side of getting a user message and sending it over the socket

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown (ucid/date must be present)

≡, Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The client prompts the user for input, reads the message, and sends it to the server through an established socket connection using an output stream.

Sub-Task

Group: Communication



Task #1: Communication

Sub Task #3: Show the code related to the Server-side receiving the message and relaying it to each connected Client

❑ Task Screenshots

4

2

1

The screenshot shows a Java IDE interface with two panes. The left pane displays a file tree with various Java files and resources. The right pane shows a code editor with Java code. A specific line of code is highlighted with a red rectangle, and a black arrow points from the text "the code related to the Server-side receiving the message and relaying it to each connected Client" to this highlighted line.

the code related to the Server-side receiving the message and relaying it to each connected Client

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown (ucid/date must be present)

≡, Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The server accepts client connections, reads incoming messages, and broadcasts these messages to all connected clients using their respective output streams.

Sub-Task

Group: Communication

Task #1: Communication

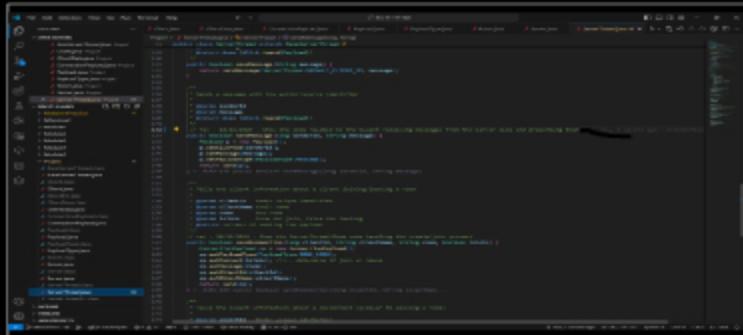
Sub Task #4: Show the code related to the Client receiving messages from the Server-side and presenting them

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1

A screenshot of a Java Integrated Development Environment (IDE) showing a code editor with several tabs open. One tab displays Java code for a client socket connection, specifically for reading messages from a server. The code includes imports for java.io and java.net packages, and uses BufferedReader to read from an input stream. Another tab shows a class definition with methods like 'onDataReceived' and 'onClientConnected'. The interface includes standard Java toolbars and a file menu.

the code related to the Client receiving messages from the Server-side and presenting them

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The client establishes a socket connection to the server, sets up an input stream to read messages, and continuously displays these messages to the user.

End of Task 1

Task

Group: Communication

Task #2: Rooms

Weight: ~50%

Points: ~1.50

100%

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.



Sub-Task

Group: Communication

Task #2: Rooms

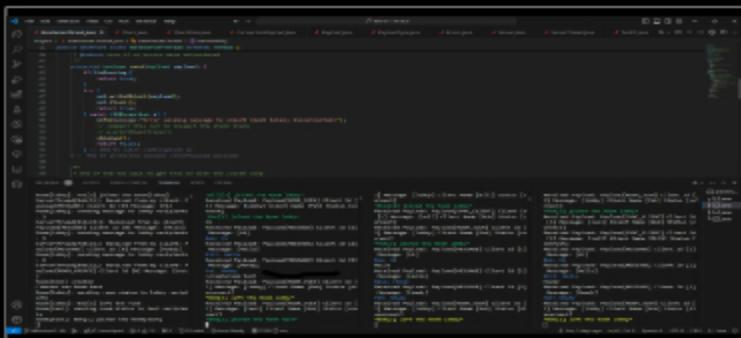
Sub Task #1: Show Clients can Create Rooms

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show Clients can Create Rooms

Caption(s) (required) ✓**Caption Hint:** *Describe/highlight what's being shown***Sub-Task**

Group: Communication

Task #2: Rooms

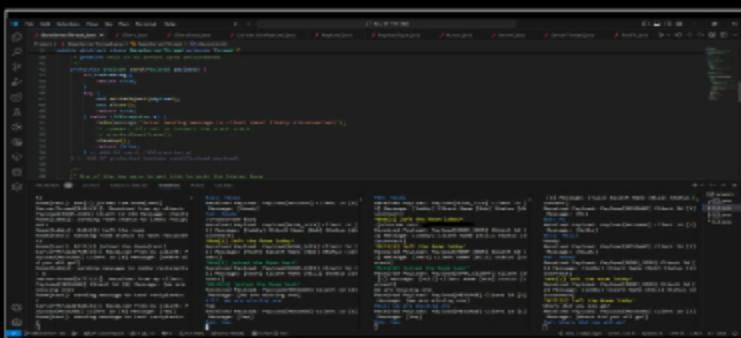
Sub Task #2: Show Clients can Join Rooms (leave/join messages should be visible)

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show Clients can Join Rooms

Caption(s) (required) ✓**Caption Hint:** *Describe/highlight what's being shown***Sub-Task**

Group: Communication

Task #2: Rooms

Sub Task #3: Show the Client code related to the create/join room commands

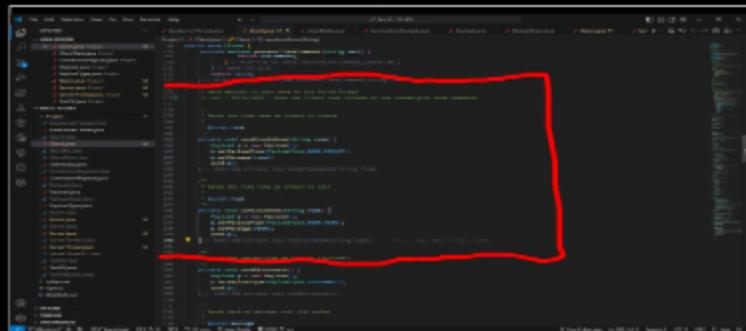
100%

Task Screenshots

Task Screenshots

Gallery Style: 2 Columns

4 2 1

A screenshot of a Java IDE (IntelliJ IDEA) showing two tabs of code. The left tab contains client-side logic for creating or joining rooms, with several methods and annotations visible. The right tab shows server-side logic, likely related to room management. A red rectangular box highlights the first few lines of the client-side code in the left tab.

Show the Client code related to the create/join room commands

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown (ucid/date must be present)

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The client sends a command to the server to create or join a room, and the server processes this command to either create a new room or add the client to an existing room, enabling group communication.

Sub-Task

Group: Communication

100%

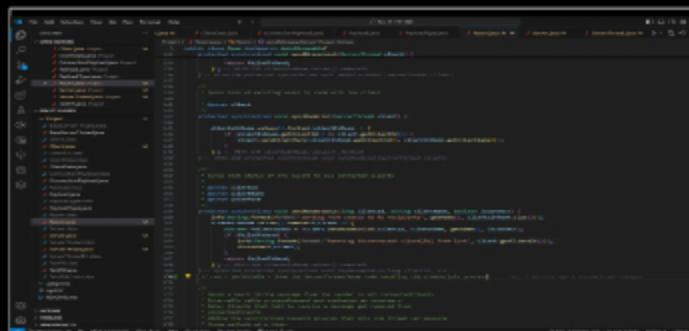
Task #2: Rooms

Sub Task #4: Show the ServerThread/Room code handling the create/join process

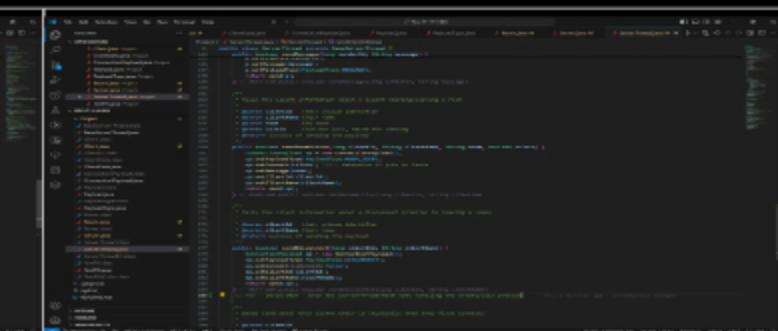
Task Screenshots

Gallery Style: 2 Columns

4 2 1

A screenshot of a Java IDE (IntelliJ IDEA) showing a single tab of code. The code is part of a class named 'Room' and contains methods for handling room creation and joining. Annotations like @Service and @Transactional are visible, indicating the code is part of a service layer.

Show the Room code handling the create/join process

A screenshot of a Java IDE (IntelliJ IDEA) showing a single tab of code. The code is part of a class named 'ServerThread' and contains logic for handling room creation and joining. It includes annotations like @Service and @Transactional, and uses reflection to handle different room types.

Show the ServerThread code handling the create/join process

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown (ucid/date must be present)

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The server thread processes client commands to create or join a room by either initializing a new room instance or adding the client to an existing room, facilitating group communication.

Sub-Task

Group: Communication



Task #2: Rooms

Sub Task #5: Show the Server code for handling the create/join process

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Show the Server code for handling the create/join process

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown (ucid/date must be present)

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The server processes create/join commands by either initializing a new room instance or adding the client to an existing room, ensuring proper room management and client allocation.

Sub-Task

Group: Communication



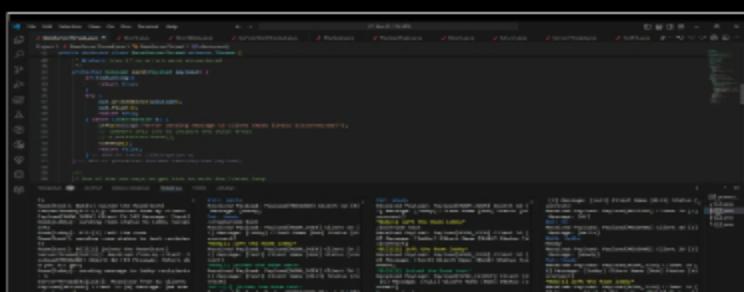
Task #2: Rooms

Sub Task #6: Show that Client messages are constrained to the Room (clients in different Rooms can't talk to each other)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show that Client messages are constrained to the Room

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain why/how it works this way

Response:

Client messages are constrained to their respective rooms because the server routes messages only to clients within the same room, ensuring isolated group communication.

End of Task 2

End of Group: Communication

Task Status: 2/2

Group

Group: Disconnecting/Termination

Tasks: 1

Points: 3

100%

▲ COLLAPSE ▲

Task

Group: Disconnecting/Termination

Task #1: Disconnecting

Weight: ~100%

Points: ~3.00

100%

▲ COLLAPSE ▲

ⓘ Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.



Columns: 1

Sub-Task

Group: Disconnecting/Termination

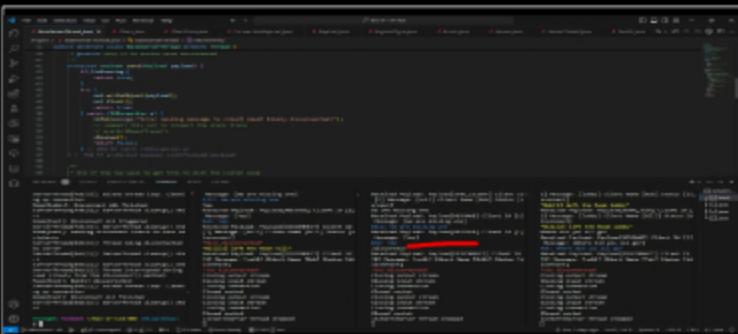
Task #1: Disconnecting

Sub Task #1: Show Clients gracefully disconnecting (should not crash Server or other Clients)

100%

Task Screenshots

Gallery Style: 2 Columns



Show Clients gracefully disconnecting

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Disconnecting/Termination

Task #1: Disconnecting

Sub Task #2: Show the code related to Clients disconnecting

Task Screenshots

Gallery Style: 2 Columns

4

Show the code related to Clients disconnecting



The screenshot shows a Java IDE interface with two panes. The left pane displays a file tree with several packages and files, including 'com.alexander.alex', 'com.alexander.alex.controllers', 'com.alexander.alex.controllers.products', 'com.alexander.alex.products', and 'com.alexander.alex.products.ProductController'. The right pane shows a code editor with Java code. A cursor is positioned at the start of a method declaration. A red box highlights the word 'get' as the code completion suggestion. The code completion dropdown menu is open, listing various options such as 'get', 'getProduct', 'getProducts', 'getProductById', 'getProductsByCategory', 'getProductsByPriceRange', 'getProductsBySearchTerm', 'getProductsBySortOrder', 'getProductsBySupplier', and 'getProductsByStatus'. The 'get' option is the first item in the list.

Show the code related to Clients disconnecting



```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.example.newsapp.NewsItem;
import com.example.newsapp.R;
import com.example.newsapp.adapters.NewsAdapter;
import com.example.newsapp.retrofit.NewsService;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class HomeFragment extends Fragment {
    private RecyclerView recyclerView;
    private NewsAdapter newsAdapter;
    private LinearLayoutManager linearLayoutManager;
    private NewsService newsService;
    private List<NewsItem> newsList;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, false);
        recyclerView = view.findViewById(R.id.recycler_view);
        linearLayoutManager = new LinearLayoutManager(getContext());
        recyclerView.setLayoutManager(linearLayoutManager);
        newsService = NewsService.getRetrofit().create(NewsService.class);
        newsList = newsService.getNews();
        newsList.enqueue(new Callback<List<NewsItem>>() {
            @Override
            public void onResponse(Call<List<NewsItem>> call, Response<List<NewsItem>> response) {
                newsAdapter = new NewsAdapter(response.body());
                recyclerView.setAdapter(newsAdapter);
            }

            @Override
            public void onFailure(Call<List<NewsItem>> call, Throwable t) {
                t.printStackTrace();
            }
        });
        return view;
    }
}
```

Show the code related to Clients disconnecting

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

When a client disconnects, the server removes the client's socket from the list of active connections and closes the associated input/output streams to free up resources.

Sub-Task

Group: Disconnecting/Termination



Task #1: Disconnecting

Sub Task #3: Show the Server terminating (Clients should be disconnected but still running)

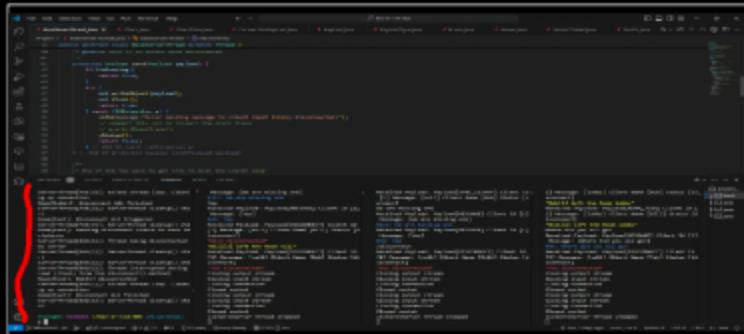
Task Screenshots

Gallery Style: 2 Columns

4 2 1

2

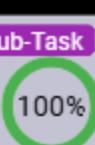
1



Show the Server terminating

Caption(s) (required)

Caption Hint: *Describe/highlight what's being shown*



Group: Disconnecting/Termination

Task #1: Disconnecting

Sub Task #4: Show the Server code related to handling termination

Task Screenshots

Gallery Style: 2 Columns

4 2 1

2

1

Show the Server code related to handling termination

Show the Server code related to handling termination

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown.*

Task

The server handles termination by closing all active client connections, releasing resources, and shutting down the application.

server socket to ensure a clean exit.

End of Task 1

End of Group: Disconnecting/Termination

Task Status: 1/1

Group

Group: Misc

Tasks: 3

Points: 1

100%

▲ COLLAPSE ▲

Task

Group: Misc

Task #1: Add the pull request link for this branch

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲

🔗 Task URLs

URL #1

<https://github.com/RicardoVas9991/Rev-IT-114-0>

URL

<https://github.com/RicardoVas9991/Rev-IT-114-0>

End of Task 1

Task

Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲

💡 Details:

Few related sentences about the Project/sockets topics



📝 Task Response Prompt

Response:

There weren't any major issues, more like having trouble finding out which code does this or that again—labeling and explaining them, too.

End of Task 2

Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▾](#)

ⓘ Details:

Grab a snippet showing the approximate time involved that clearly shows your repository.

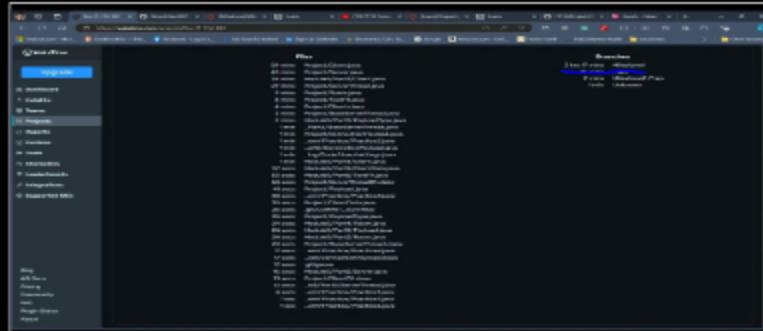
The duration isn't considered for grading, but there should be some time involved.



▣ Task Screenshots

Gallery Style: 2 Columns

4 2 1



WakaTime Screenshot



WakaTime Screenshot

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment