**Unloop Full-Stack Web Dev**

***Building a TCP Chat Server in Node.js: Part 1***

#### Reference List:

-Node Up and Running (O’Reily): Path: R:\Full Stack Web Development\Readings\Digital Books

#### File List:

-client folder, containing client.js and package.json

#### Software List:

-Git Bash

-Node.js

# Introduction

You are about ready to create your very own TCP chat server. The transmission control protocol (TCP) is commonly used for connection-oriented tasks, such as chat rooms, because it maintains a reliable session between the client and server.

So, what is a chat server and how does it work?

A chat server facilitates the conversation of several clients who are connected to the server. In the simplest explanation, the server echoes the messages of a client to all other clients. The server is the focal point of communication, that is, clients don’t actually talk to each other directly—they must go through the server in order to talk to each other.

This requires two parts:

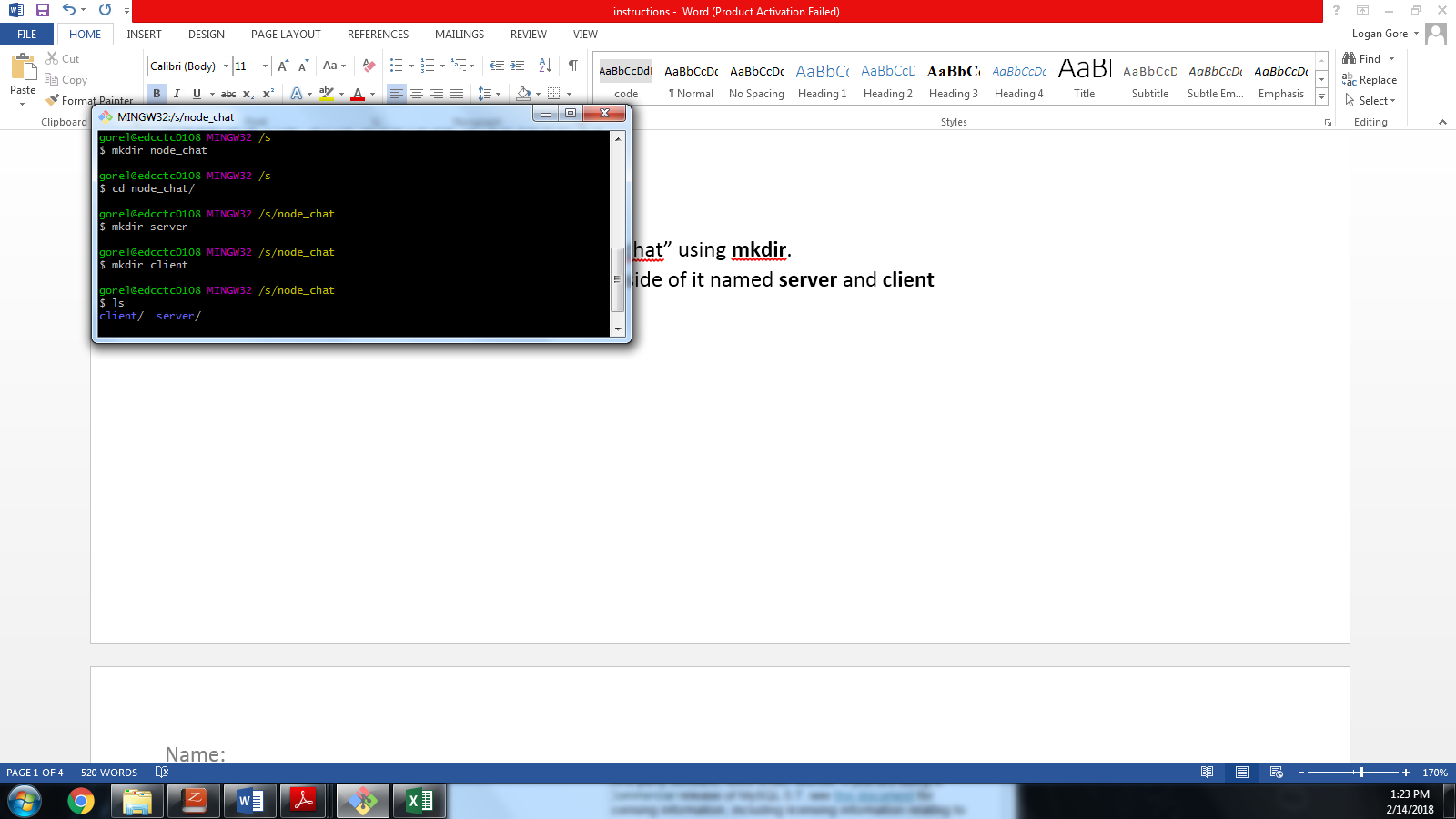
1. The server, which accepts client connections, receives messages from these clients, and then broadcasts a client message to other clients bounces the messages around; and
2. The client, which sends their messages to the server, and receives other clients’ messages from the server.

# To illustrate how these parts normally operate as separate entities, we are going to create them and run them as two separate node.js projects.

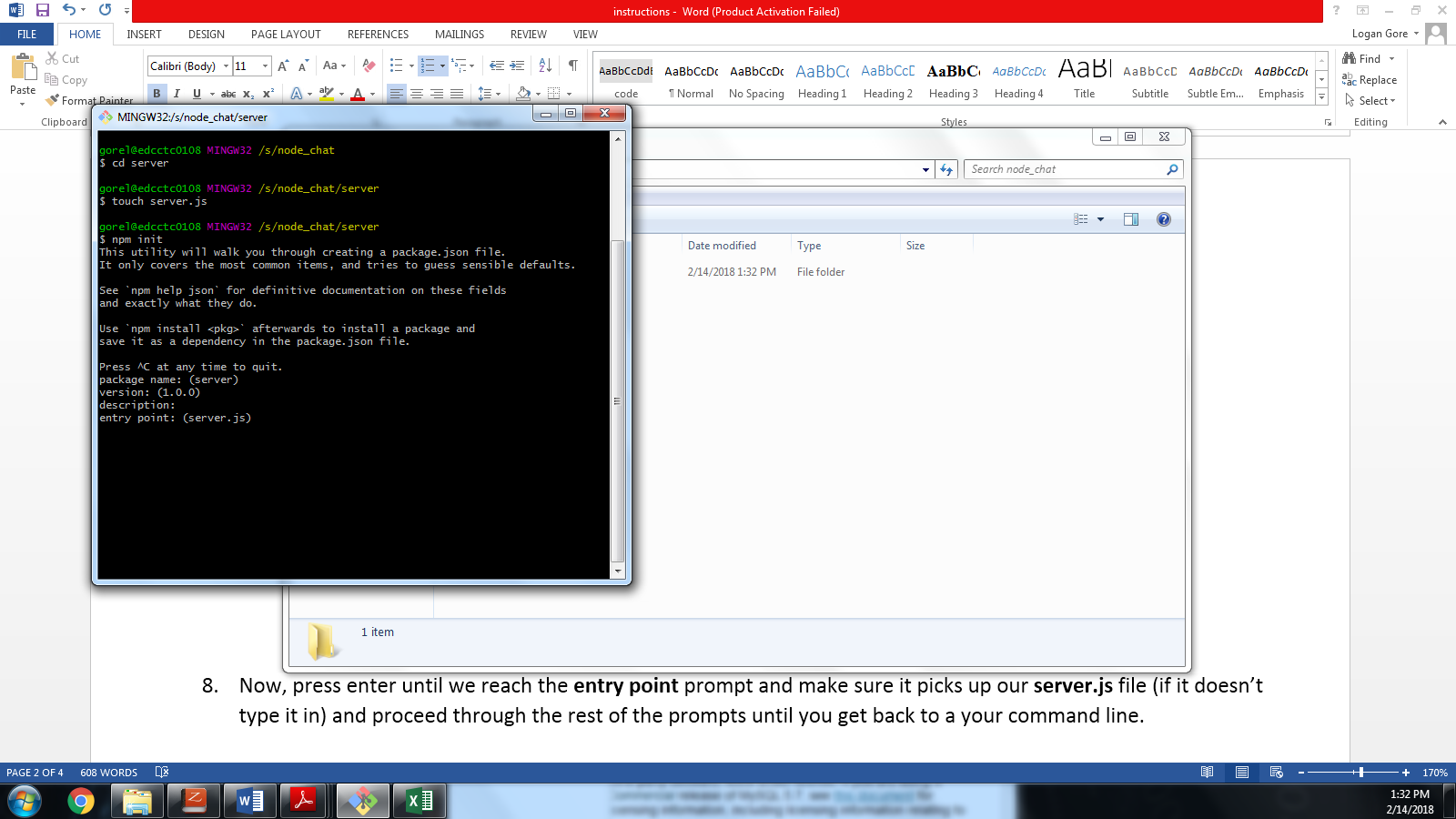
# Setting up our project directories

First, we need to set up our Node.js project:

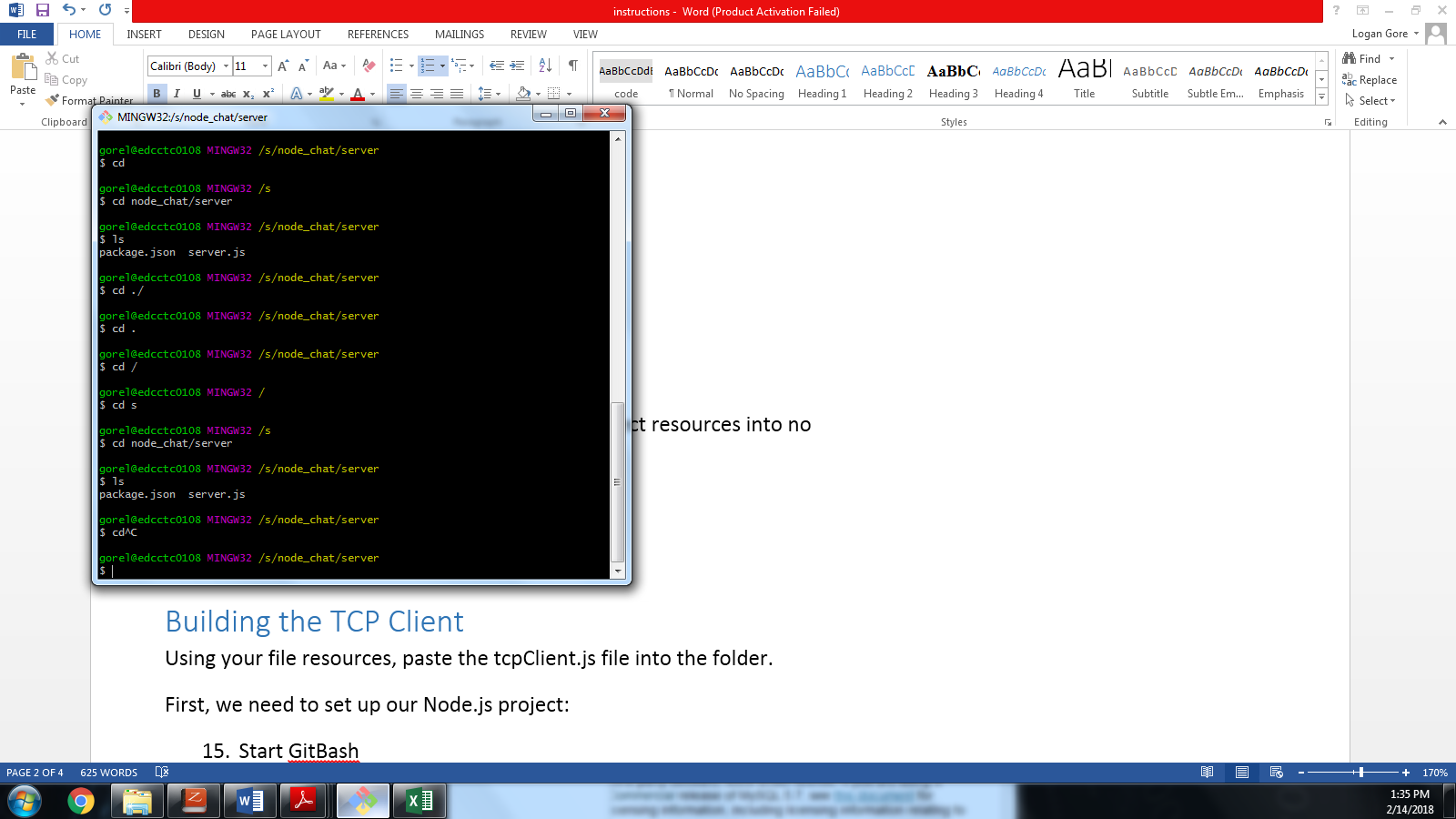
1. Start Git Bash
2. Create folder in your drive named “node\_chat” using **mkdir**.
3. **cd** to this folder, and create a folder inside of it named **server**



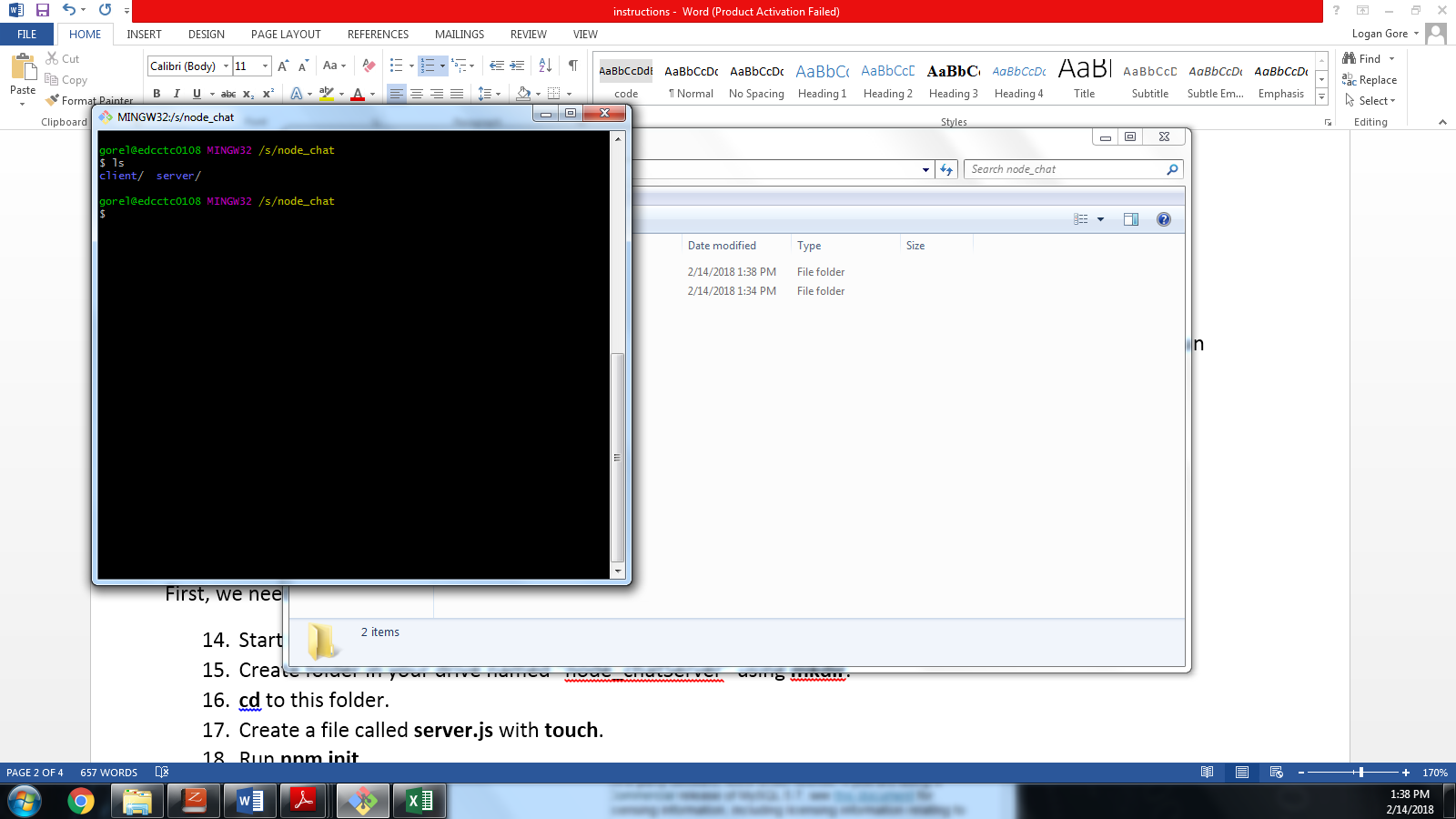
1. Verify with **ls**
2. **cd** to the **server** folder.
3. Create a file called **server.js** with **touch**.
4. Run **npm init**
5. Now, press enter until we reach the **entry point** prompt and make sure it picks up our **server.js** file (if it doesn’t type it in) and proceed through the rest of the prompts until you get back to a your command line.



1. Our **server** project is now ready to go!
2. Run ls to check that you have a package.json file and your server.js file.



1. Now, **cd ..** back to your **node\_chat** folder
2. Copy the **client** folder from the project resources into the **node\_chat** folder. The client folder contains an existing client project, similar to the server project we just created.
3. Run **ls** to make sure our **node\_chat** folder contains both the **client** and **server** folders.

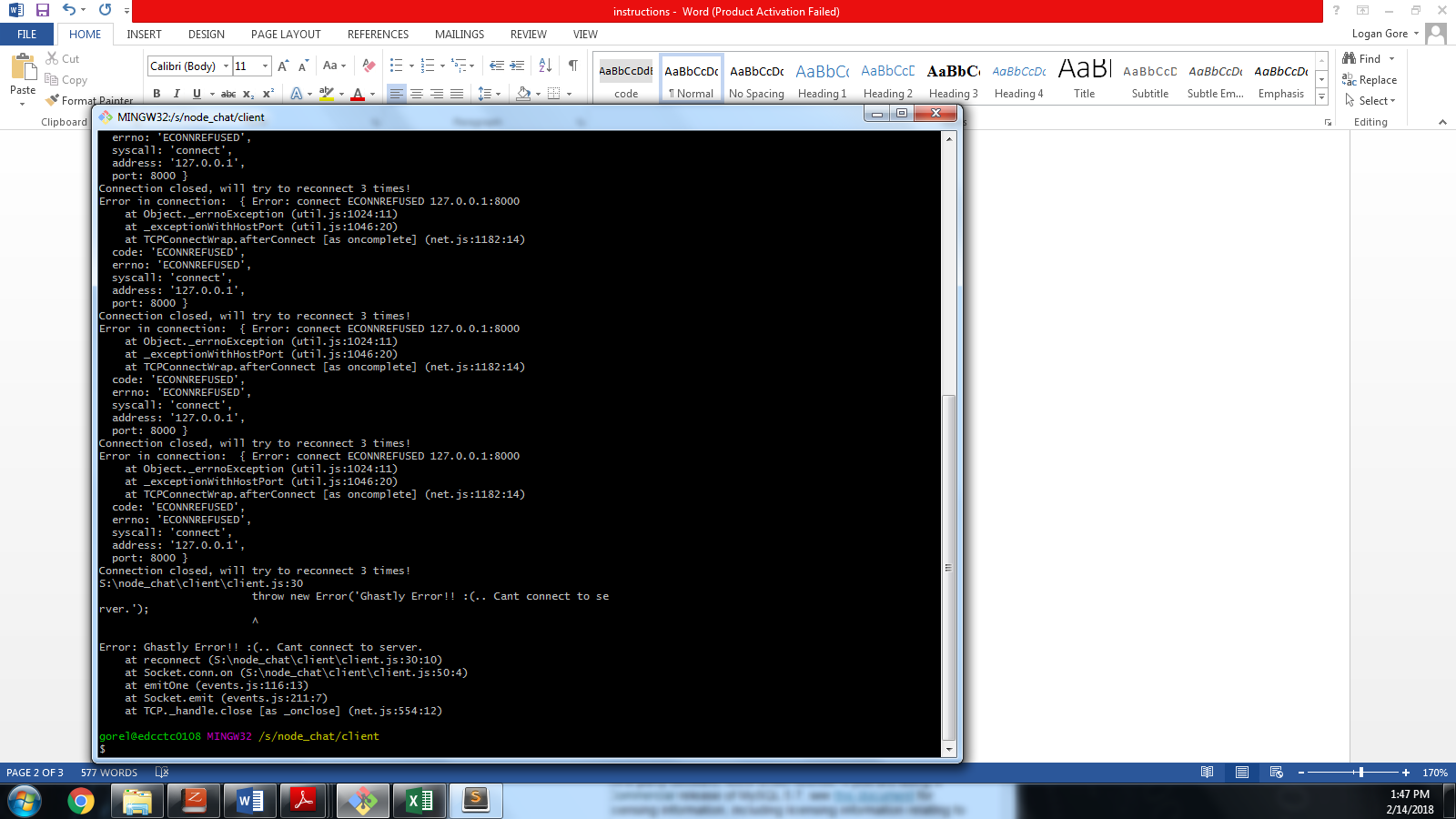


Let’s starting coding!

# Running the TCP Client

**cd** to your client folder and run **node client.js**

You should get a nice long error message that ends with something like:



This means our client can’t find a server to connect to.

If it found a server that was listening, it would have established a connection. We’ll be using the TCP client in the following “Building the TCP Server” Section

# Building the TCP Server

Now we’re ready to start writing code!

Open the **Node Up and Running** book and go to *Chapter 2 Doing Interesting Things* (page 15).

Your task is to follow the directions, writing the code in your **server.js** file.

\*\*\*This example will ask you to test your code using a Telnet client. BUT we don’t have a Telnet client… and instead we’re going to use the TCP client you pasted into your **client** folder.

At this point you should read the article titled “Simple Questions, What is Telnet” in the same location as the assignment instructions.

After reading up to example 2-1 in the book, type in the code in your server.js file and save it:

*Example 2-1. Creating a new TCP server*

var net = require('net')

var chatServer = net.createServer()

chatServer.on('connection', function(client) {

client.write('Hi!\n');

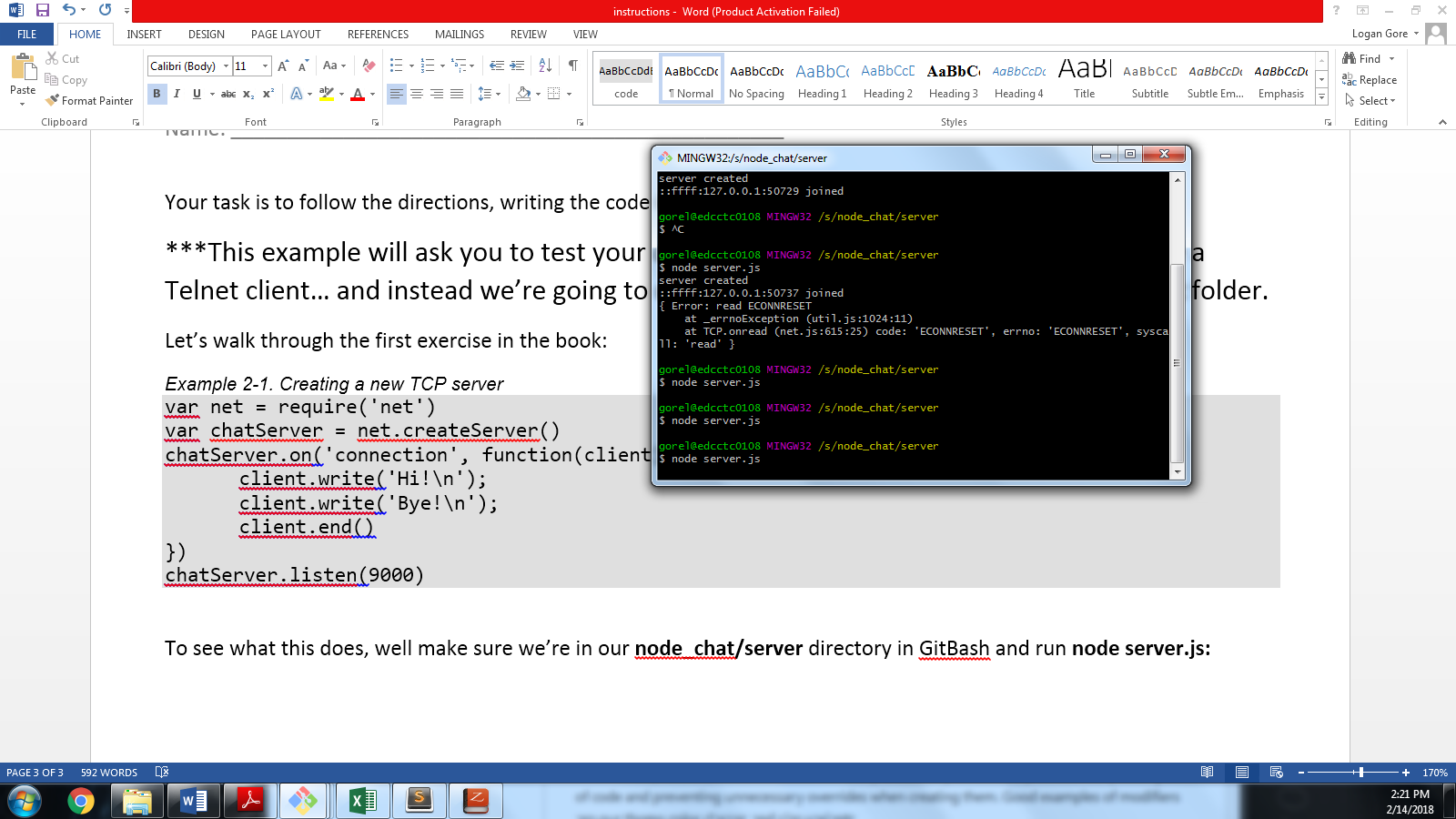
client.write('Bye!\n');

client.end()

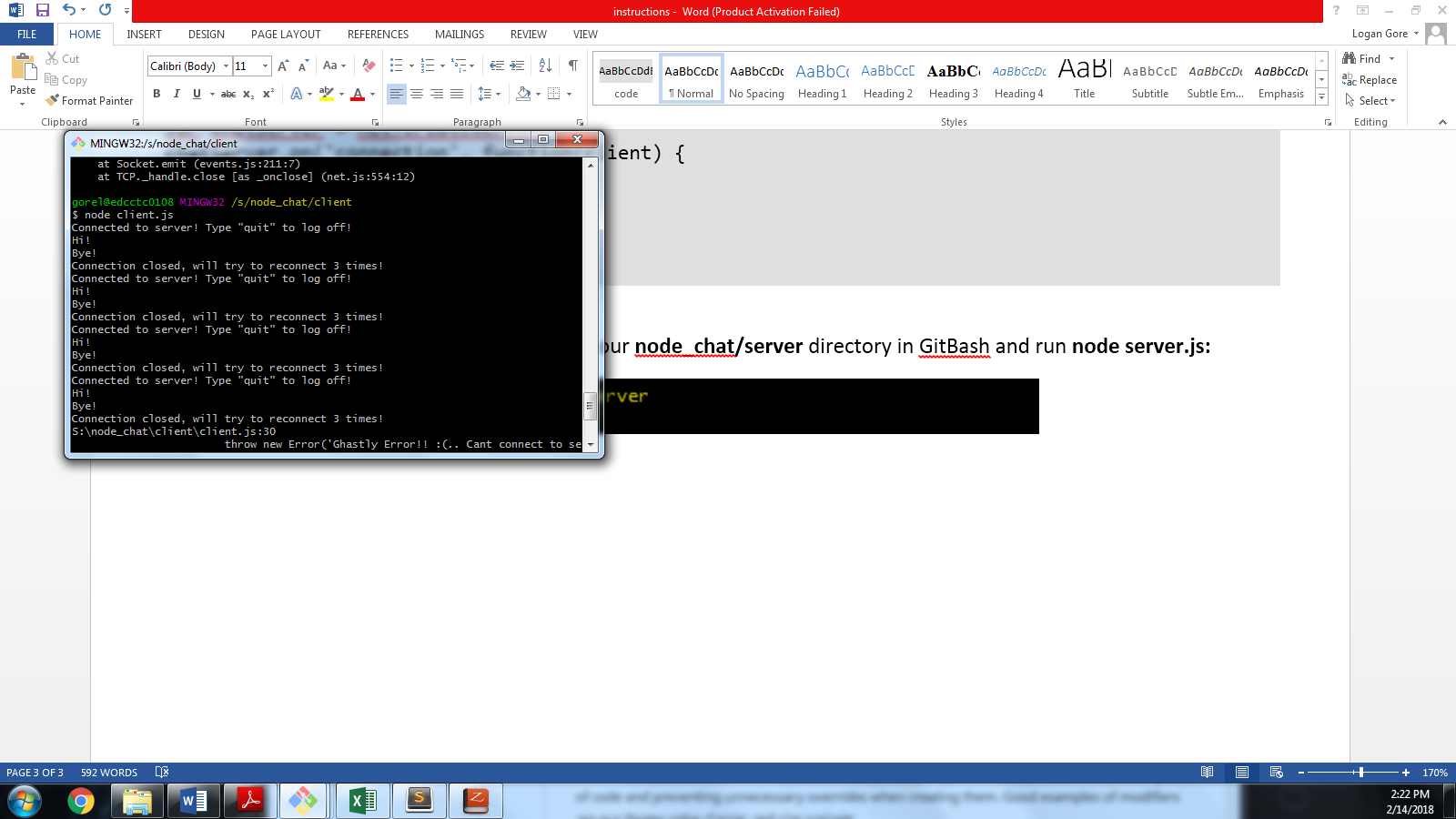
})

chatServer.listen(9000)

To see what this does, well make sure we’re in our **node\_chat/server** directory in Git Bash and run **node server.js:**



And…it doesn’t really do anything, but we’ll see if it’s actually listening by using running our client from **node\_chat/client** by typing **node client.js** in a separate bash shell:



We can see that the server actually connected, said “Hi” and “Bye” and then closed its connection. As you can see, our client will always try to connect three times.

**Okay, so now you can go through the rest of the examples, running the client.js instead of Telnet.**

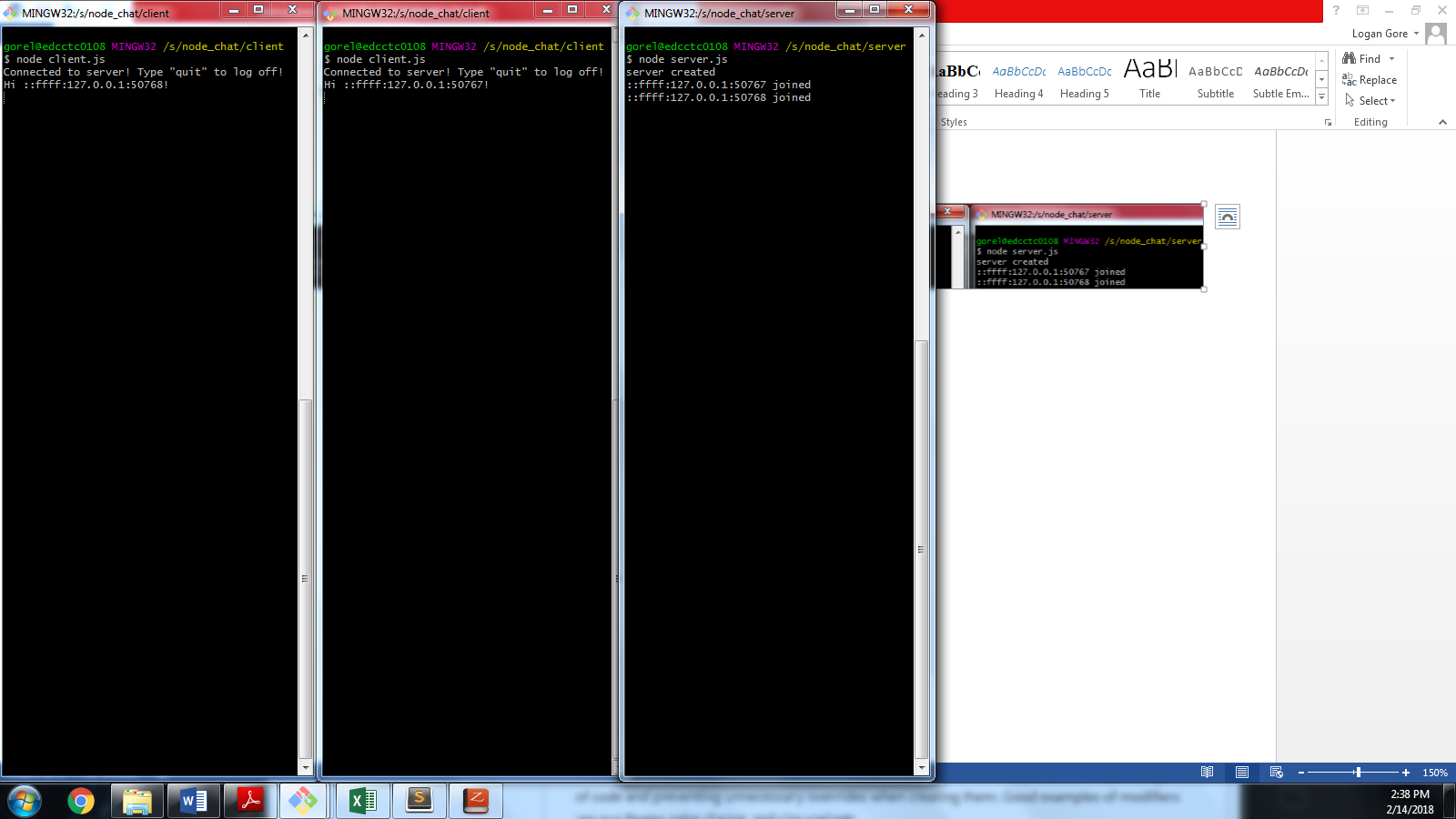
**Make sure you always start your server first, so the client has something to connect to and doesn’t error out.**

# Testing your chat client and server:

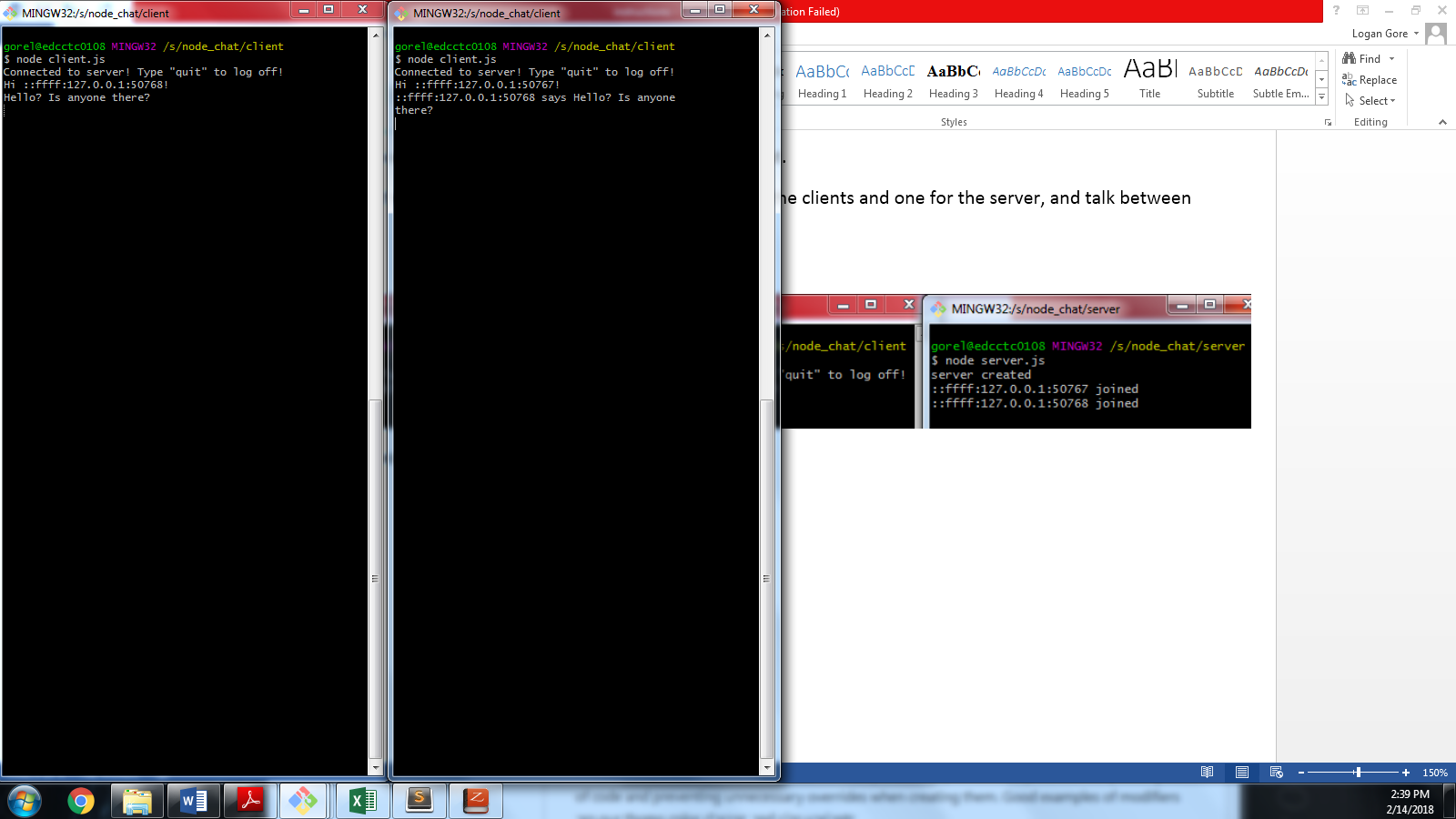
By the end of the exercise (page 23) you should have a working chat.

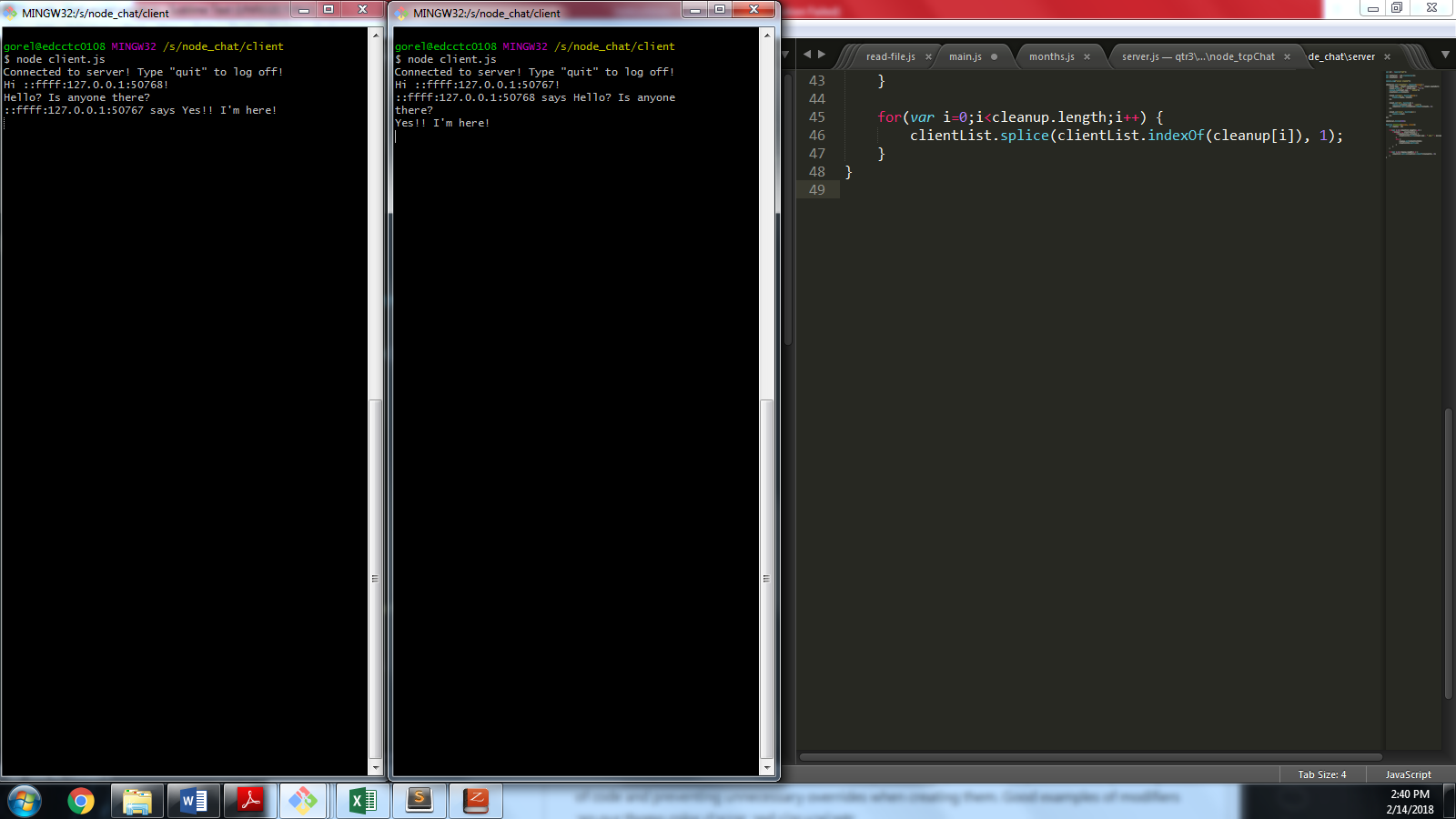
If you want to see the chat “in action” open up two bash shells for the clients and one for the server, and talk between the clients.

Like so:



If we type some messages in the clients, the server will echo them:





# Challenge: Modifying the Server

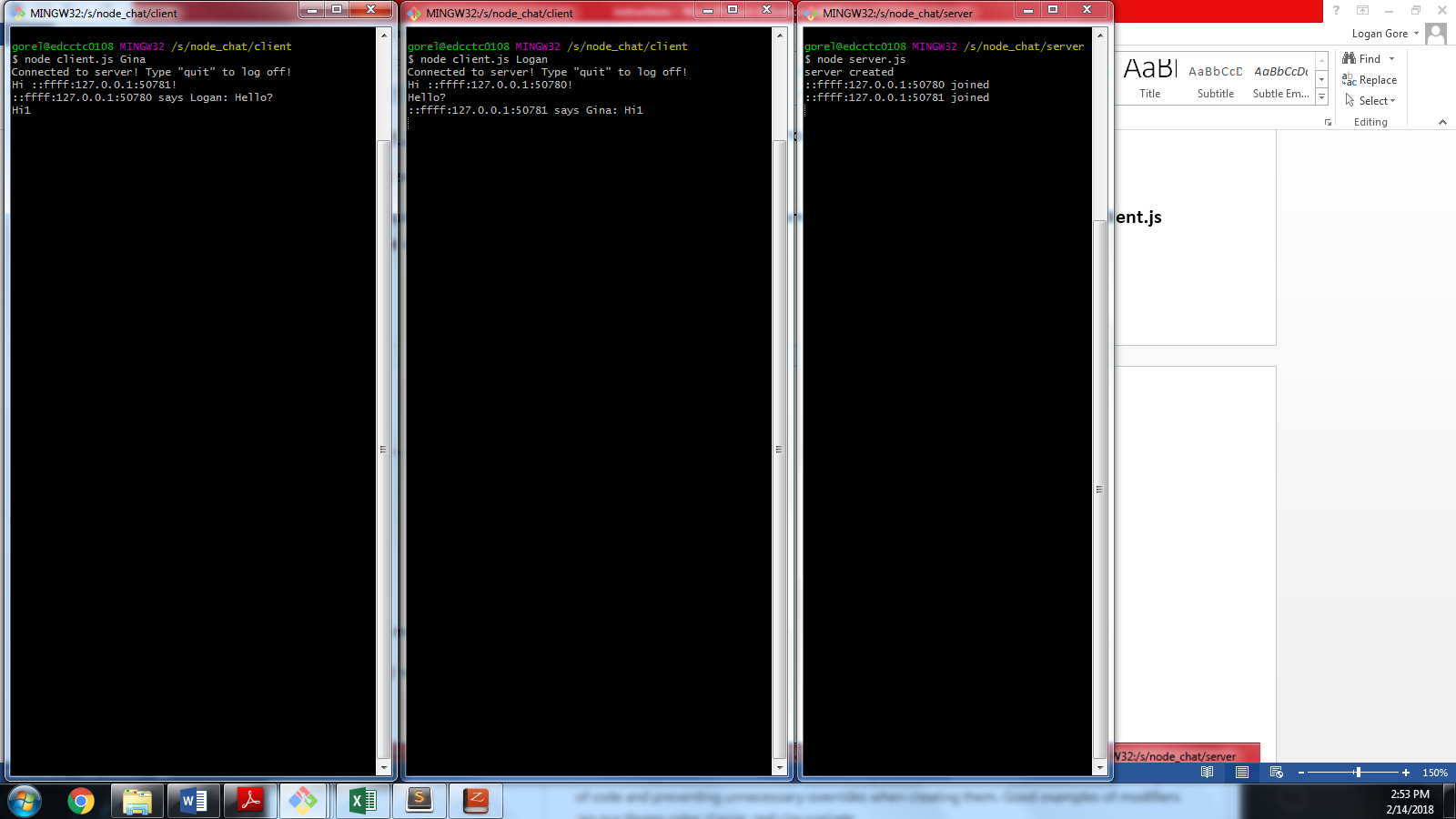
You’ll notice that the **::FF:127.0**…. is not a very chatty naming set up for a chat server.

Instead we could do something else…

An easy solution would be to use a person’s name in the second parameter of the client process i.e. “**node client.js Logan**”.

Actually, the client is already modified for this to work!

Let’s see…Does it work?



Nope…the numbers are still there along with the names…Even worse!

Your challenge is to modify the server, to digest this new info.

Here’s what the end result should behave like:

Hint: You should not need to write any code to accomplish this. You should only need to remove/comment out lines!

