Assignment 8: Rust async/await

Question 1: Consider the following attempt to model a simple server:

```
1   use std::io;
2   use tokio::time::{sleep, Duration};
3   use chrono::Utc;
7
8   use std::collections::HashMap;
9
10  async fn handle_cmd(cmd: char) {
11      match cmd {
12          ‘i’ => {
13              if let Ok(r) = reqwest::get(“https://httpbin.org/ip”).await {
14                  if let Ok(ip) = r.json::<HashMap<String,String>>().await {
15                      println!(“{:?}”, ip);
16                  }
17              }
18          },
19          ‘d’ => {
20              if let Ok(r) = reqwest::get(“https://httpbin.org/delay/9”)
.await {
21                 println!(“{:#?}”, r);
22              }
23          },
24          other => println!(“unknown cmd: {}”, other),
25       }
26  }
27
28  async fn read_cmd() -> Result<usize, Box<dyn std::error::Error>> {
29      let mut data = String::new();
30      let stdin = in::stdin();
31      match stdin.read_line(&mut data) {
32          Ok(len) => {
33              if len > 0 {
34                  if let Some(cmd) = data.chars().next() {
36                      handle_cmd(cmd).await;
37                  }
38              }
39              Ok(len)
40          },
41          Err(e) => Err(Box::new(e)),
42      }
43  }
44
46  async fn periodic_task(interval: u64) {
47      sleep(Duration::from_secs(interval)).await;
48      println!(“UTC now: {}”, Utc::now());
49  }
50
51  #[tokio::main]
52  async fn main() -> Result<(), Box<dyn std::error::Error>> {
53      loop {
54          match read_cmd().await {
```

```
                Ok(len => if len == 0 {
                    break;
                },
                Err(e) => println!("input error {}", e),
            };
            periodic_task(5).await;
        }

        Ok(())
    }

```

You can see that the implementer was modeling an event-driven server that accepted input requests and serviced them by requesting some action from a back-end provider. In addition, the server is required to perform a periodic task, which happens to be printing out the current time every interval seconds.

The code above is available at: https://github.com/uofa-ece421/sync-server
If you "cargo run" it you will see that the functionality is correct, but the server is not actually concurrent.

Your assignment is to make the server actually work as intended. Note that you cannot use threads to make the server concurrent, you must stick with async tasks. Also note that you will likely have to refactor the loops so the tasks can run concurrently.