ChatGPT: Send & Receive https with Rust

Here’s a basic example of a **Rust application** that can send and receive HTTPS requests using the popular **reqwest** and **tokio** libraries. This example demonstrates how to:

1. Send an HTTPS GET request to an API.
2. Handle the response.
3. (Optionally) Create a basic HTTP server to receive requests.

Step 1: Add Dependencies

Add the following dependencies to your Cargo.toml:

```
[dependencies]
reqwest = { version = "0.11", features = ["json", "tls"] }
tokio = { version = "1", features = ["full"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
```

Step 2: Application Code

Here’s a complete Rust application that sends and receives HTTPS requests:

```
use reqwest::Client;
use serde::{Deserialize, Serialize};
use tokio::time::{sleep, Duration};
use std::error::Error;

// Struct to deserialize JSON responses
#[derive(Serialize, Deserialize, Debug)]
struct ApiResponse {
  message: String,
}
```

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn Error>> {
    // Create an HTTP client
    let client = Client::new();

    // Example 1: Sending an HTTPS GET request
    let api_url = "https://jsonplaceholder.typicode.com/posts/1";
    println!("Sending GET request to: {}", api_url);

    let response = client
        .get(api_url)
        .send()
        .await?
        .json::<serde_json::Value>()
        .await?;

    println!("GET response: {}", response);

    // Example 2: Sending an HTTPS POST request
    let post_url = "https://jsonplaceholder.typicode.com/posts";
    println!("Sending POST request to: {}", post_url);

    let request_body = serde_json::json!({
        "title": "foo",
        "body": "bar",
        "userId": 1
    });

    let post_response = client
```

```
        .post(post_url)
        .json(&request_body)
        .send()
        .await?
        .json::<serde_json::Value>()
        .await?;

    println!("POST response: {}", post_response);

    // Example 3: Basic HTTP server to receive requests
    tokio::spawn(async {
        if let Err(e) = run_server().await {
            eprintln!("Server error: {}", e);
        }
    });

    // Keep the main program running while server handles requests
    println!("Server running on http://localhost:8080");
    sleep(Duration::from_secs(10)).await;

    Ok(())
}

// A simple HTTP server to receive requests
async fn run_server() -> Result<(), Box<dyn Error>> {
    use warp::Filter;

    // Define a simple route
    let route = warp::path!("hello" / String).map(|name: String| {
```

```
    format!("Hello, {}!", name)
  });

  // Start the server
  warp::serve(route).run(([127, 0, 0, 1], 8080)).await;
  Ok(())
}
```

Explanation of the Code

1. **Sending HTTPS GET and POST Requests:**
    - The reqwest::Client is used to send requests.
    - The GET request retrieves data from an example API (https://jsonplaceholder.typicode.com).
    - The POST request sends a JSON payload and receives a response.
2. **Server Implementation:**
    - The warp crate is used to set up a simple HTTP server.
    - The server listens on http://localhost:8080 and responds with a greeting when accessed at http://localhost:8080/hello/{name}.
3. **Asynchronous Execution:**
    - The tokio runtime handles asynchronous operations, allowing the program to send requests and run a server concurrently.

Running the Application

1. **Install the Required Dependencies:** Ensure your Cargo.toml has the listed dependencies.
2. **Build and Run:**
3. cargo run
4. **Test the Server:**
    - Open your browser or use curl to test the server:
    - curl http://localhost:8080/hello/Rustacean
5. **Test API Requests:** The program logs the responses from the example API to the console.

This example demonstrates sending HTTPS requests and running an HTTP server. Let me know if you’d like to expand on this, such as adding error handling or other features! 😊