

Leo De Silva

A Level Computer Science

MINI PROJECT: SPACETRADERS

A TUI client for the SpaceTraders API

2024, St Albans School

Contents

1	Analysis	2
1.1	Programming Language	2
1.2	API	5
2	Design	6
3	Technical Solution	6
4	Testing	6
5	Evaluation	6

1 Analysis

A web API (Application Programming Interface) is a set of standard protocols to interact with a web server. This project aims to produce a user-friendly means to interact with the SpaceTraders API.

1.1 Programming Language

The first decision to make regarding to this project is that of programming language – more specifically a choice between two stable, statically typed languages with strong module support and asynchronous runtimes. Go and Rust. The former (renowned for its simplicity and concurrency) would offer an expedited development cycle with an extensive standard library. Rust, however, offers memory safety, performance and a strict typing system – enforcing good programming practices. In addition to a superior multiplatform bundler for distributing the program as a single binary without dependencies. Furthermore, Rust has a more mature and documented ecosystem for Terminal User Interfaces (TUI) and Command Line Interfaces (CLI) – which I intend to explore as a means for the user to interact with the API. Below is the same program demonstrating registration with the SpaceTraders API written in Rust and Go respectively to illustrate their differences.

```
1 use std::collections::HashMap;
2 use request::{Client, Error};
3
4 async fn register() -> Result<String, Error> {
5     let client = Client::new();
6
7     let agent = HashMap::from([
8         ("symbol", "L30_DESILVA"),
9         ("faction", "COSMIC")
10    ]);
11
12    let res = client
13        .post("https://api.spacetraders.io/v2/register")
14        .header("Content-Type", "application/json")
15        .json(&agent)
16        .send()
17        .await?
18        .json::<serde_json::Value>()
19        .await?;
20
21    if let Some(error) = res.get("error") {
22        println!("{}", error["message"]);
23    } else {
24        println!("Congratulations, {}. You have been
25            registered. Please note your token.",
26            res["data"]["agent"]["symbol"]);
```

```

27     println!("{}", res["data"]["token"]);
28 }
29
30 Ok(res["data"]["token"].to_string());
31 }
32
33 #[tokio::main]
34 async fn main() {
35     let _ = register().await.unwrap();
36 }

```

```

1 $ http_prototype git:(master): cargo run
2     Compiling http_prototype v0.1.0
3     Finished dev [unoptimized] target(s) in 0.51s
4     Running 'target/debug/http_prototype'
5
6 Congratulations, "L30_DESILVA". You have been registered.
7 Please note your token:
8 "eyJhbGciOiJIc2E6dECLg"
9
10 $ http_prototype git:(master): cargo run
11     Finished dev [unoptimized] target(s) in 0.07s
12     Running 'target/debug/http_prototype'
13
14 "Cannot register agent. Agent symbol L30_DESILVA has already
    ↪ been claimed."

```

The equivalent Go code, due to its simplicity, can be tedious and lengthy to write.

```

1 package main
2
3 import (
4     "bytes"
5     "encoding/json"
6     "fmt"
7     "net/http"
8 )
9
10 type Register struct {
11     Data map[string]interface{}
12     Error map[string]interface{}
13 }
14

```

```

15 func main() {
16     body, err := json.Marshal(map[string]string{
17         "symbol": "TRUCKER",
18         "faction": "COSMIC",
19     })
20
21     if err != nil {
22         panic(err)
23     }
24
25     req, err := http.NewRequest(
26         "POST",
27         "https://api.spacetraders.io/v2/register",
28         bytes.NewBuffer(body)
29     )
30
31     if err != nil {
32         panic(err)
33     }
34
35     req.Header.Add("Content-Type", "application/json")
36     client := &http.Client{}
37     res, err := client.Do(req)
38     if err != nil {
39         panic(err);
40     }
41
42     defer res.Body.Close()
43
44     data := Register{}
45     err = json.NewDecoder(res.Body).Decode(&data)
46     if err != nil {
47         panic(err)
48     }
49
50     if len(data.Error) != 0 {
51         fmt.Println(data.Error["message"])
52     } else {
53         var agent map[string]interface{}
54         agentJson, err := json.Marshal(data.Data);
55         if err != nil {
56             panic(err)
57         }
58
59         err = json.Unmarshal(agentJson, &agent);
60         if err != nil {

```

```

61         panic(err)
62     }
63
64     fmt.Printf("Congratulations, You've been successfully
        ↪ registered. Please note your access token:\n")
65     fmt.Println(agent["token"])
66 }
67 }

```

Due to its stability, performance, security and mature, centralised ecosystem I will use rust for this project. Although Rust's somewhat convoluted approach to asynchronous programming will require careful design considerations as to not introduce bugs.

1.2 API

The Space Traders API offers HTTP end points with which programs can access, and in turn: play the open universe trading game. The breadth of available features ranges from locating all available shipyards in a system:

```

1    curl 'https://api.spacetraders.io/v2/systems/:
    ↪ systemSymbol/waypoints?traits=SHIPYARD' --header '
    ↪ Authorization: Bearer INSERT\_TOKEN\_HERE '

```

To selling ship cargo:

```

1    curl --request POST \
2    --url 'https://api.spacetraders.io/v2/my/ships/:
    ↪ miningShipSymbol/sell'
3    --header 'Authorization: Bearer INSERT_TOKEN_HERE' \
4    --header 'Content-Type: application/json' \
5    --data '{
6        "symbol": "IRON_ORE",
7        "units": "100"
8    }'

```

However, there are 2 versions of the SpaceTraders API in production, the complete V1, and the alpha V2. The former is simpler, more mature, and yet lacks much of the functionality of the second release. Whereas V2 consists of a larger universe, a wider breadth of features, and comprehensive documentation. Thus, my project will utilise the V2 API specification albeit hesitant of potential bugs due to the alpha nature of its release.

- 2 Design**
- 3 Technical Solution**
- 4 Testing**
- 5 Evaluation**