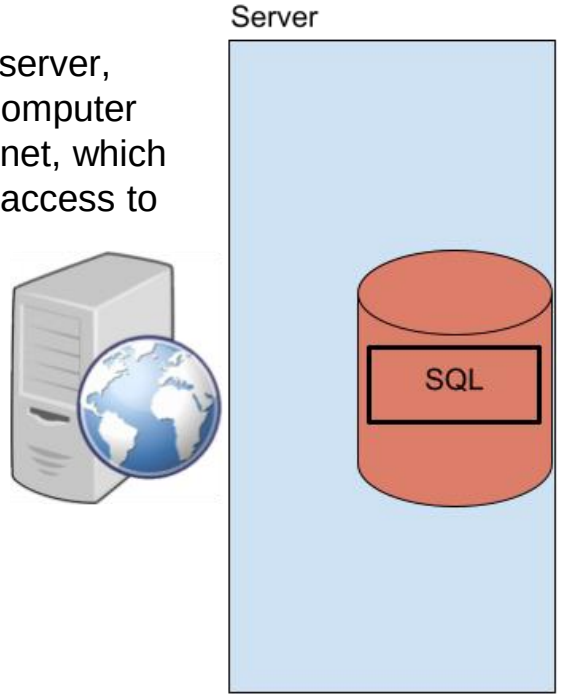
API Review Day

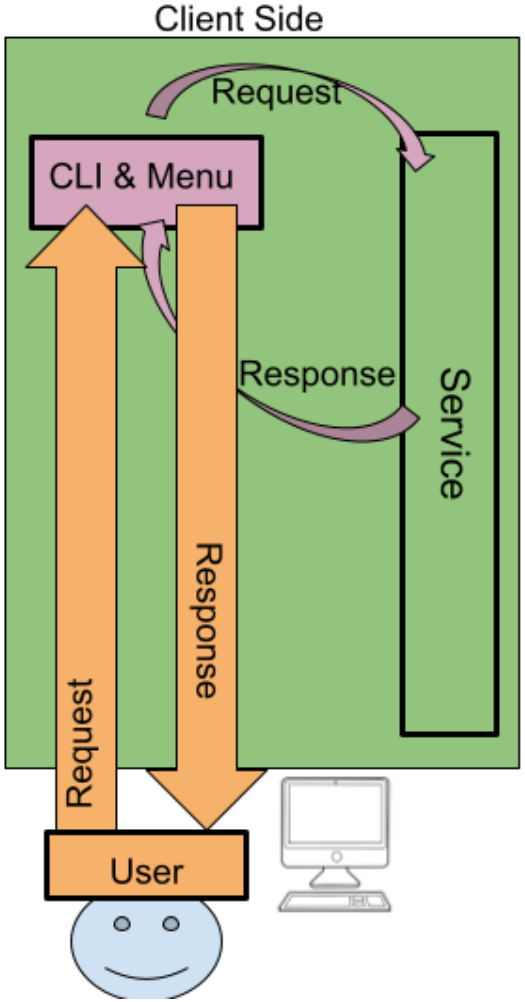
Module 2: 15

The problem: how can the User interact with the database?

The data (SQL) is on a server, which is on a different computer somewhere on the internet, which the user does not have access to use.



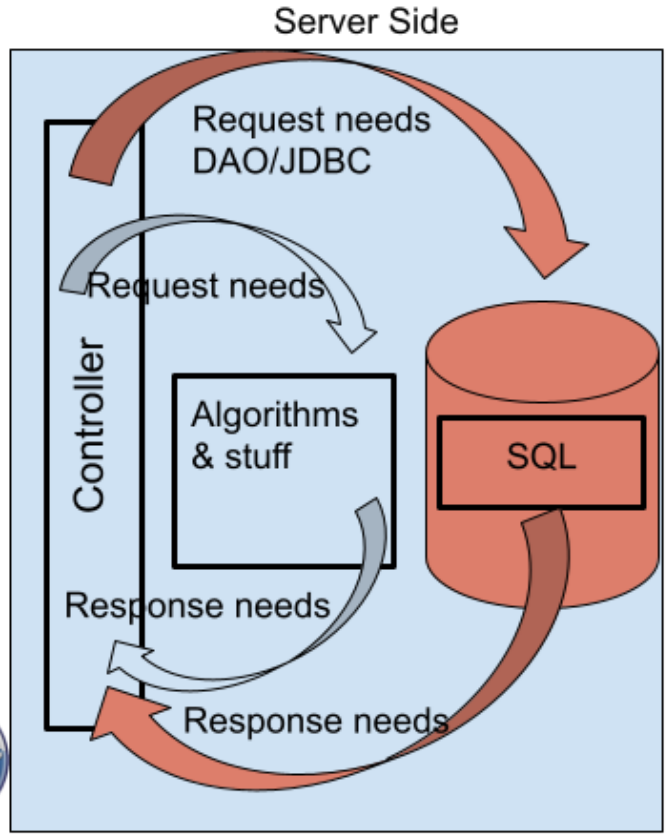
The User is using their device, which may be a computer of any internet enabled device, like their phone.

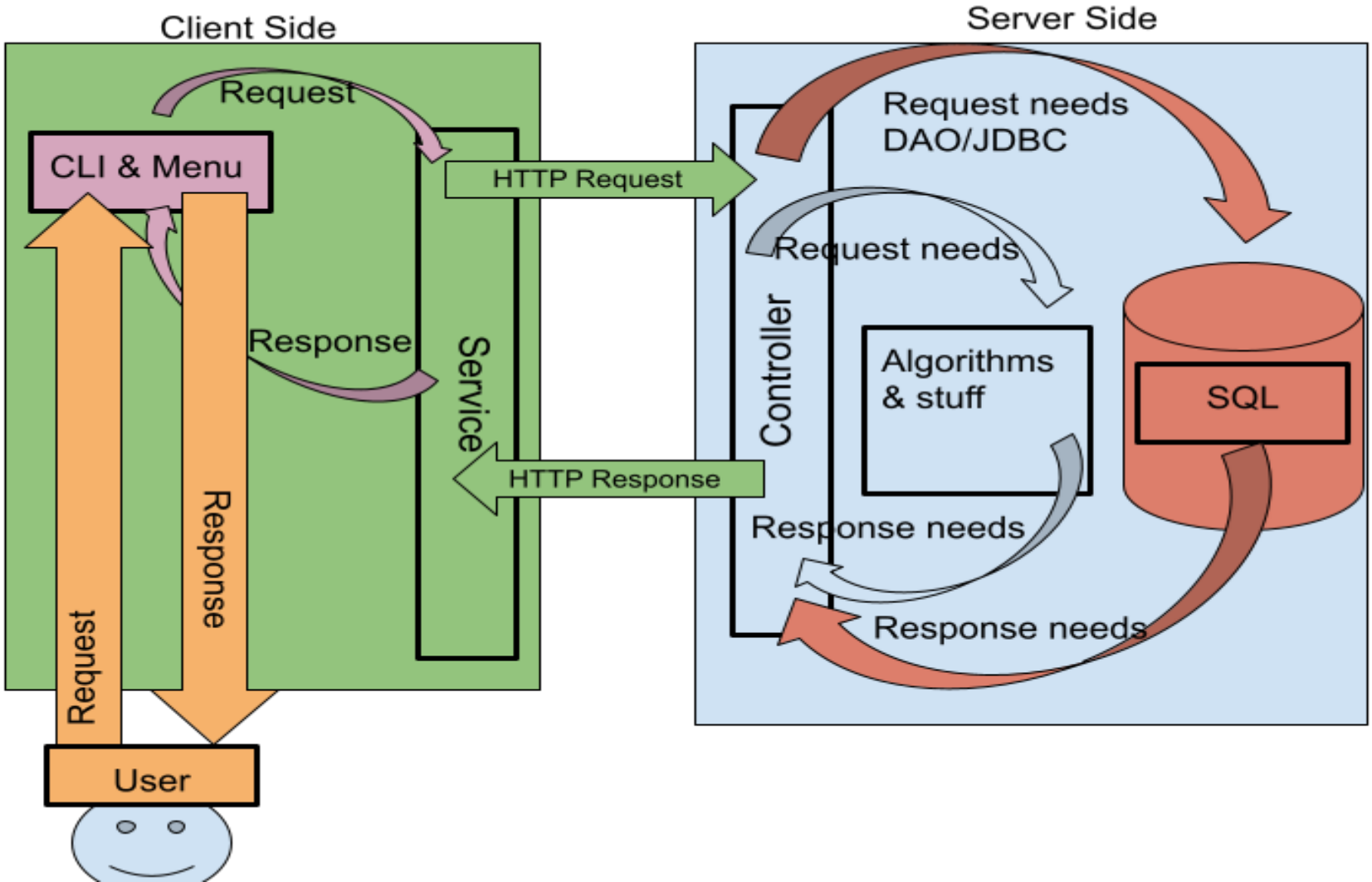


Solution

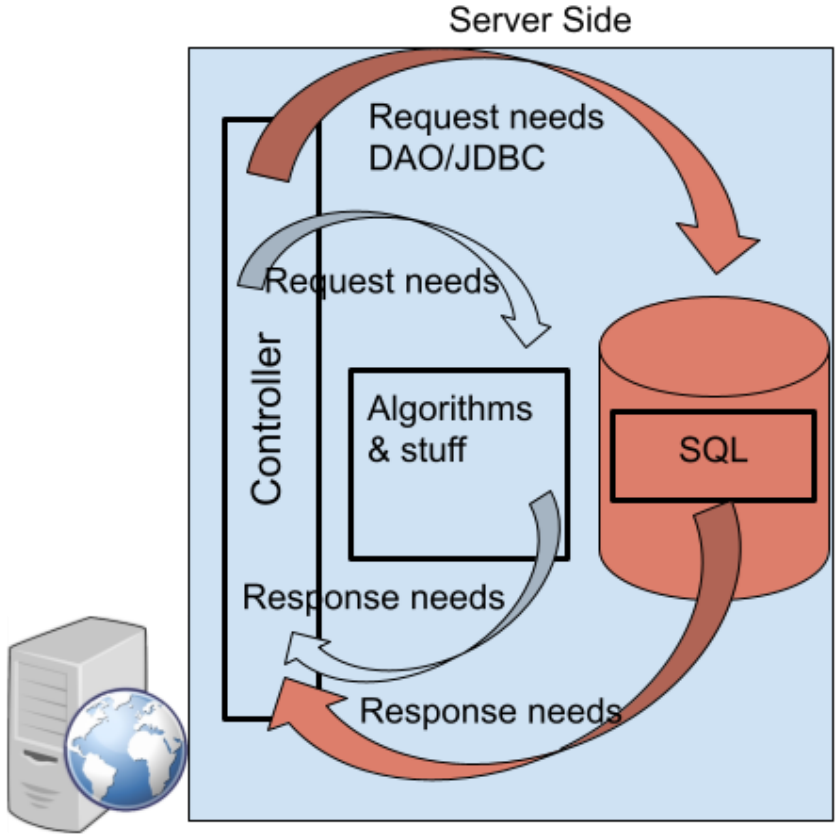
An application on the User's device can communicate with the Server (another computer) on the internet using HTTP.

A RESTful API on the Server can allow this communication.




Client Side
Server Side
Request
CLI & Menu
HTTP Request
Request needs
DAO/JDBC
Request needs
Response
Service
Controller
Algorithms
& stuff
SQL
HTTP Response
Response needs
Response needs
Request
Response
User

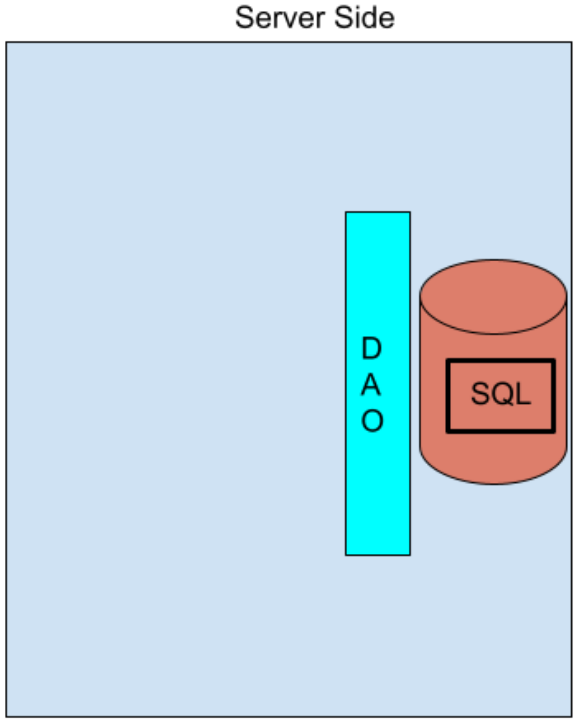
The Server


Server Side
Request needs
DAO/JDBC
Request needs
Controller
Algorithms
& stuff
SQL
Response needs
Response needs

1) Encapsulate the Database using the DAO Pattern

Build DAOs the encapsulate the Database functionality.

The DAO will be injected into the controller by dependency injection, so the JDBCDao class needs the `@Component` annotation and to accept a `JdbcTemplate` as a constructor argument.



Test the DAO using Integration Testing before continuing!

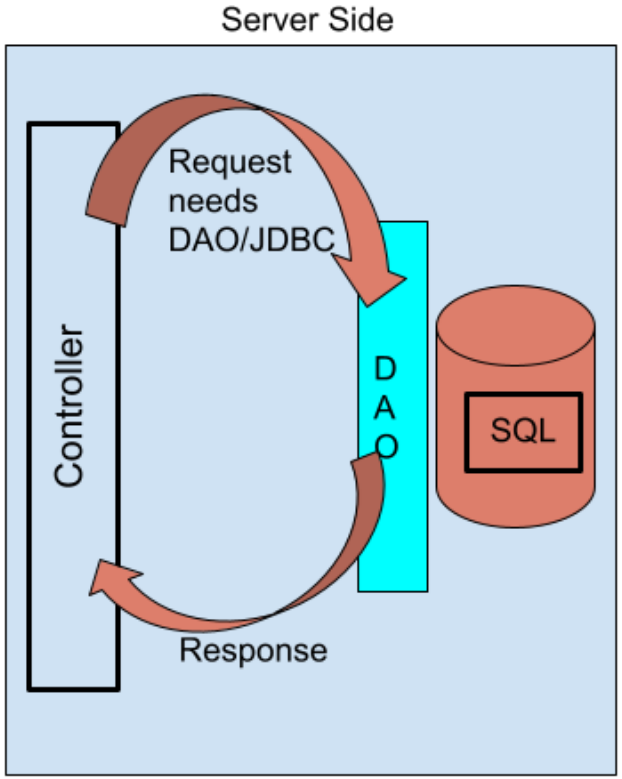
2) Add a Controller to expose an Endpoint for the data

Add a Controller (@RestController) to expose an API endpoint (@RequestMapping) for the user to work with the data in the database.

The Controller will have access to the DAO using dependency injection. The controller method from the endpoint will call the DAO to retrieve data from the database and return it serialized as JSON.

If non-database algorithmic code needs to be run, then the controller may also call those classes.

Test the controller using Postman before continuing!



The Client / Server Connection

The Connection between the client and the server is the API endpoint.

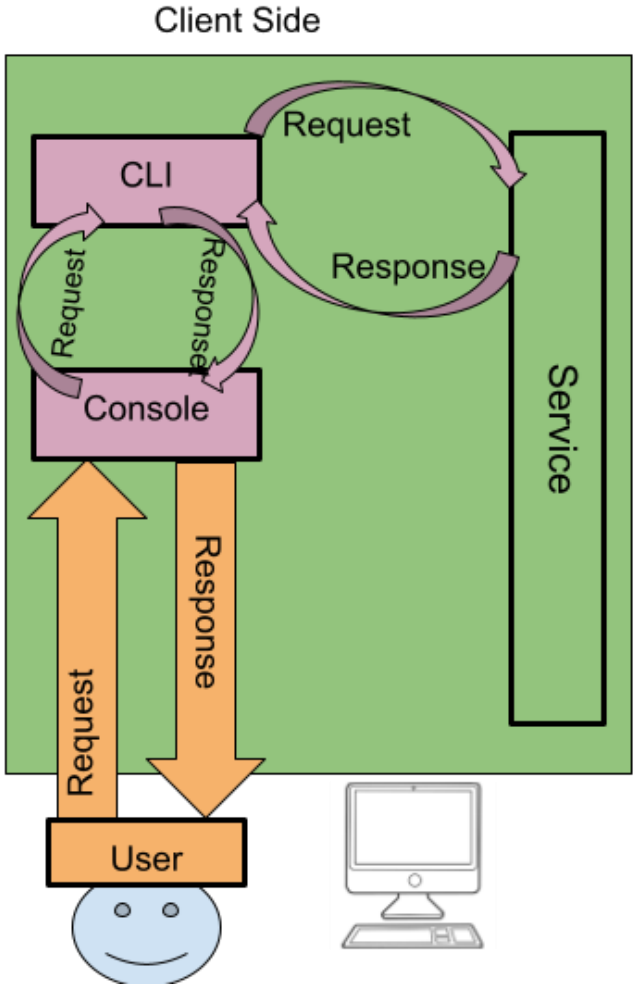
```
http://localhost:8080/addressbook
```

The Clients sends a Request to the Server using the Endpoint and the HTTP Application Protocol and gets a response as an HTTP Status Code and JSON

Both the Client and Server need their own data objects to serialize/deserialize the JSON message. The properties of these data objects must match the key names in the JSON. However, not all keys need to be present in the data object.


```
[
    {
        "contactId": 1,
        "firstName": "Dave",
        "lastName": "Warthog",
        "dateAdded": "2021-06-24"
    },
     {
        "contactId": 4,
        "firstName": "Jenny",
        "lastName": "Jones",
        "dateAdded": "2021-06-24"
    },
    {
        "contactId": 6,
        "firstName": "Jung",
        "lastName": "Choi",
        "dateAdded": "2021-06-24"
    }
]
```

The Client

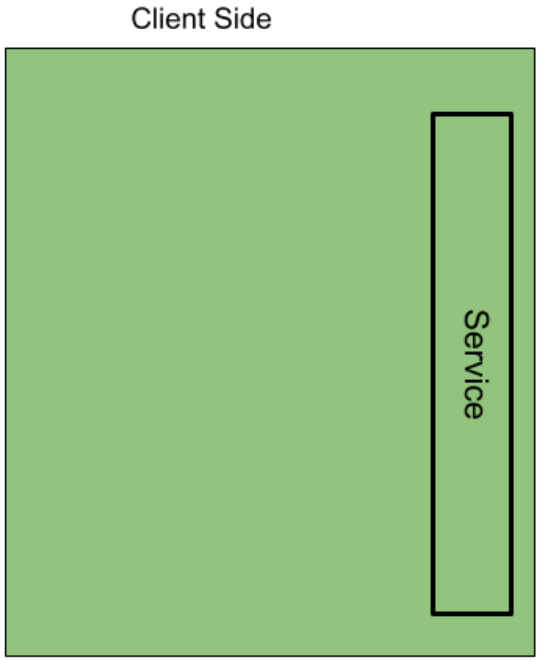

Client Side
Request
CLI
Response
Request
Response
Console
Service
Request
Response
User

1) Build a Service to call the API

Build a Service class to encapsulate the API.

The service will use the RestTemplate to call the API endpoints and deserialize the results.

A data object that matches the JSON response will be needed.

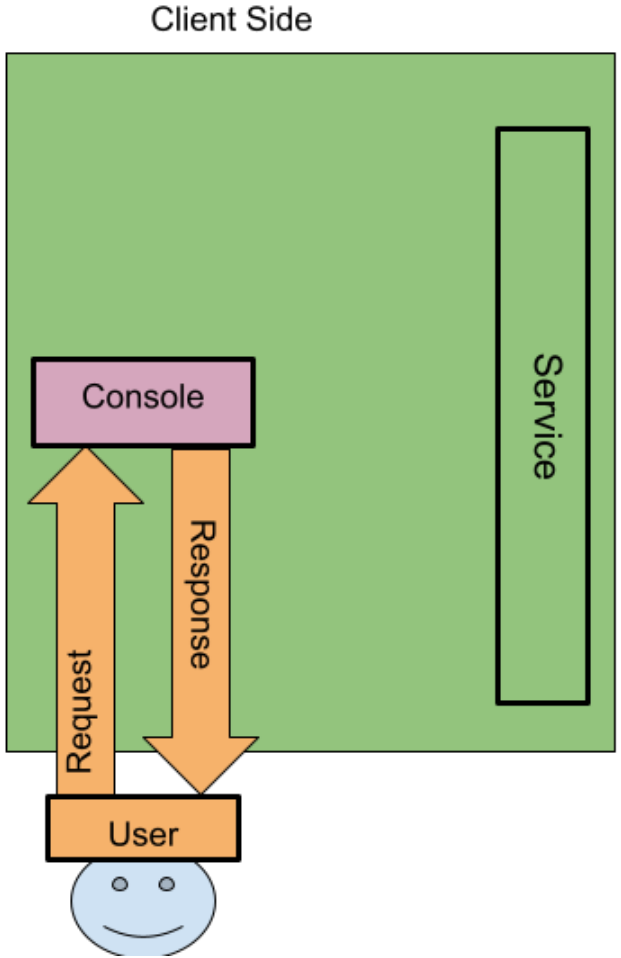


2) Build the Console Service

The ConsoleService will take input from the User and print responses for them.

It should accept the data needed as arguments to its methods and return values the user selects as those methods return types.

The ConsoleService will make no decisions and do no processing, it will only accept/give user input/output.



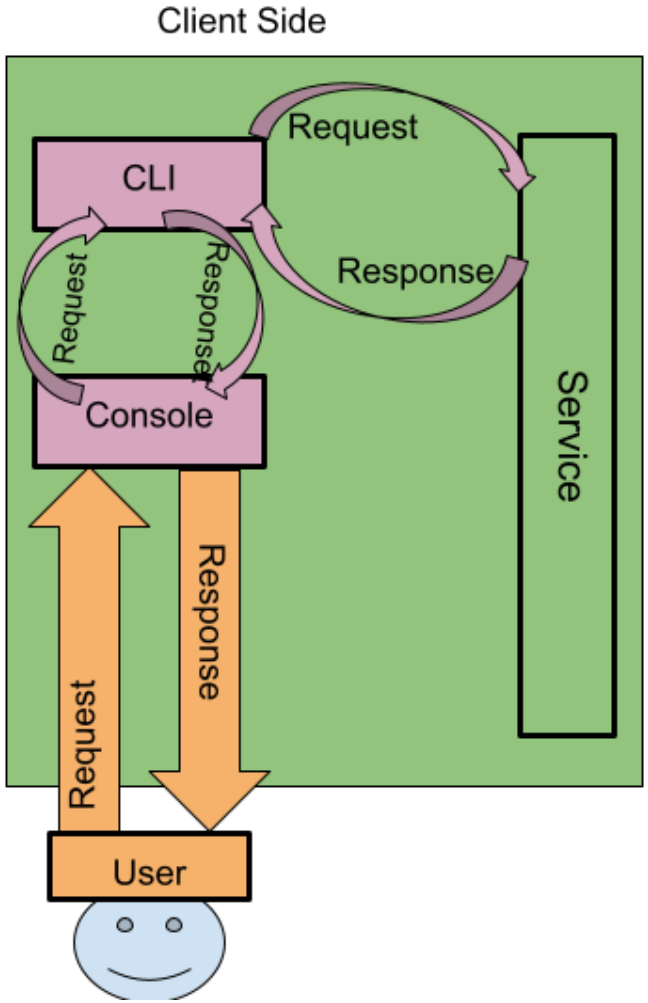
2) Build the CLI

Build the CLI.

The CLI should use the API Service to get data from or send data to the API.

Once the CLI has data from the API Service it will use the ConsoleService to display that data to the user.

The CLI will also use the ConsoleService to ask the user what they want to do next, and then make decisions on how to process that request.



Final Workflow