



UNIVERSITY OF  
RICHMOND

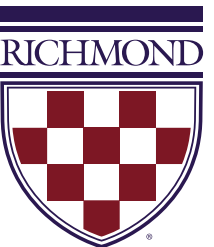
# CMSC 240 Lecture 17

**CMSC 240 Software Systems Development**  
Fall 2023



# Today – REST APIs

- Project Introduction
- REST APIs
- In-Class Exercises



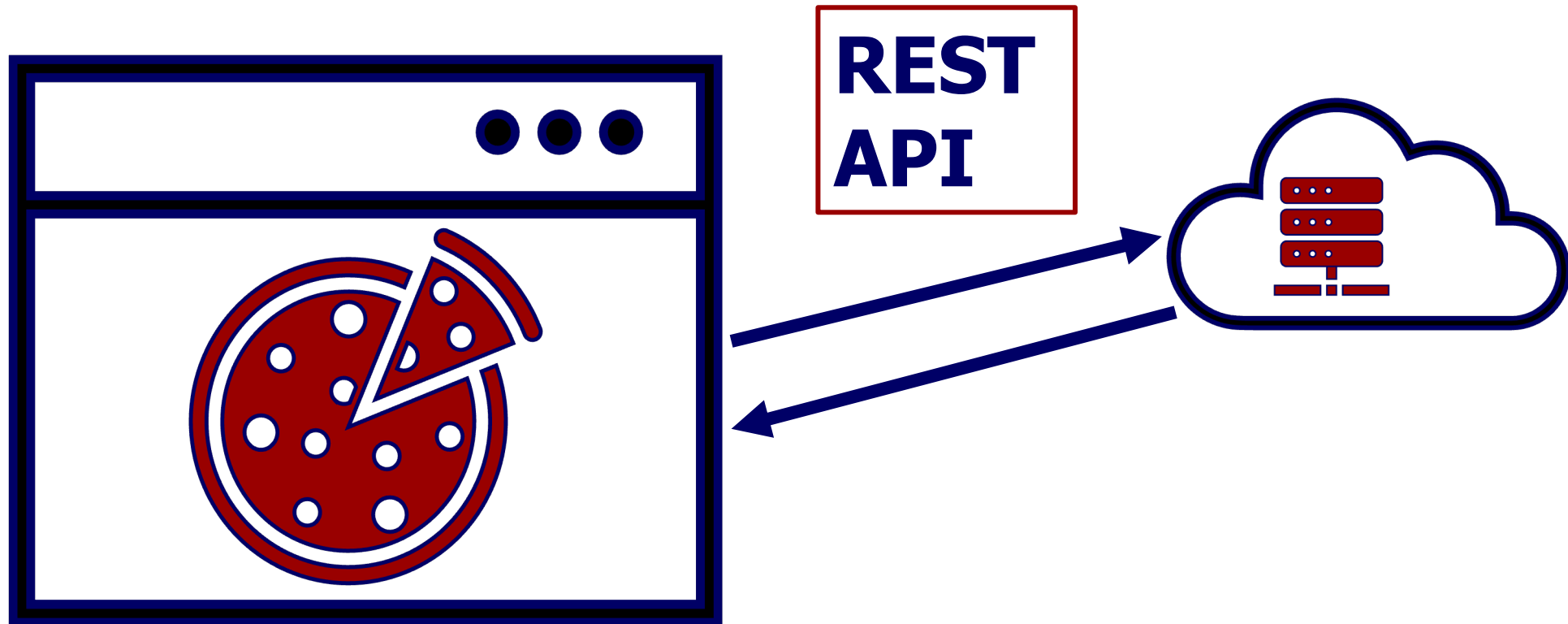
# Project

Build a custom web service of your own design

1. Create self selected teams of size 2
2. Propose an idea for a web service
3. Create a design document including UML
4. Implement your web service in C++
5. Add a unit testing suite

# REST API

- **R**epresentational **S**tate **T**ransfer
  - Communication between client and server “It’s how they talk”
  - “RESTful” web service

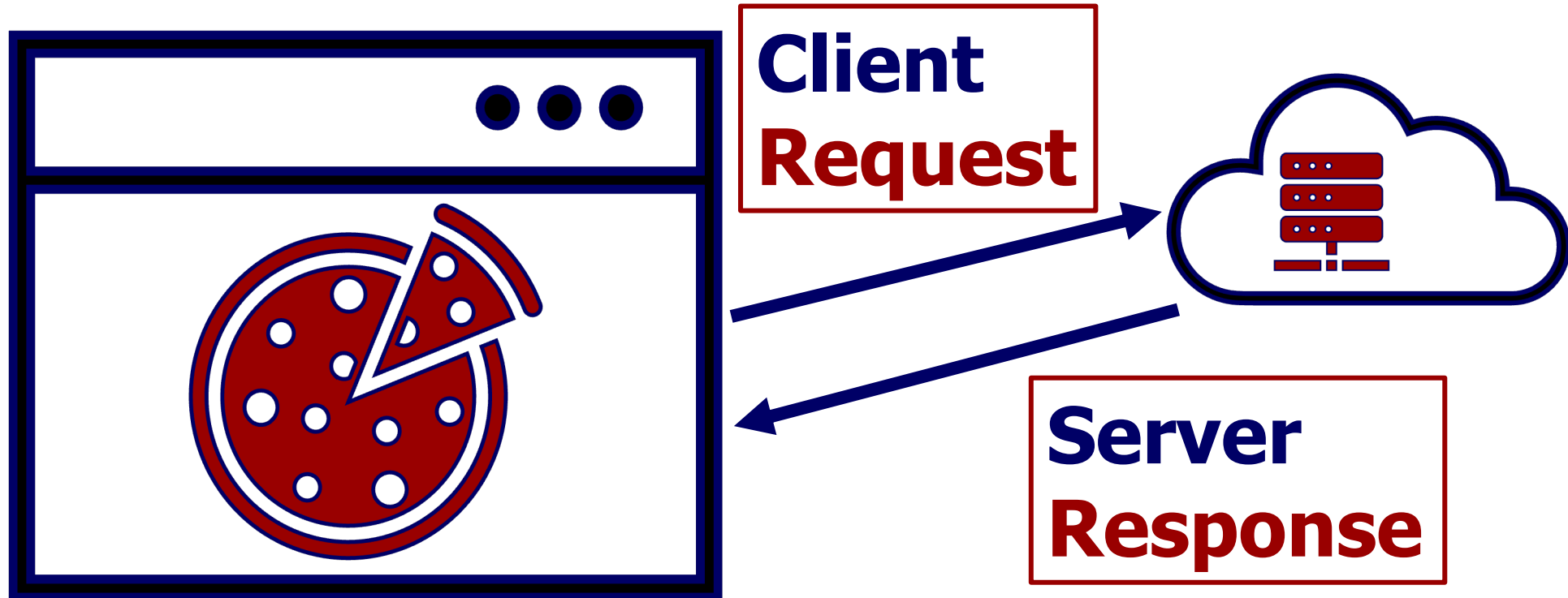


# REST API

- Benefits of REST
  - Simple
  - Standardized
  - Scalable
  - Stateless
  - High Performance

<http://urpizza.com/api/toppings>

**Resource**



# Request

What actions (verbs) would you want to perform on your resource?

## HTTP Methods/Operations

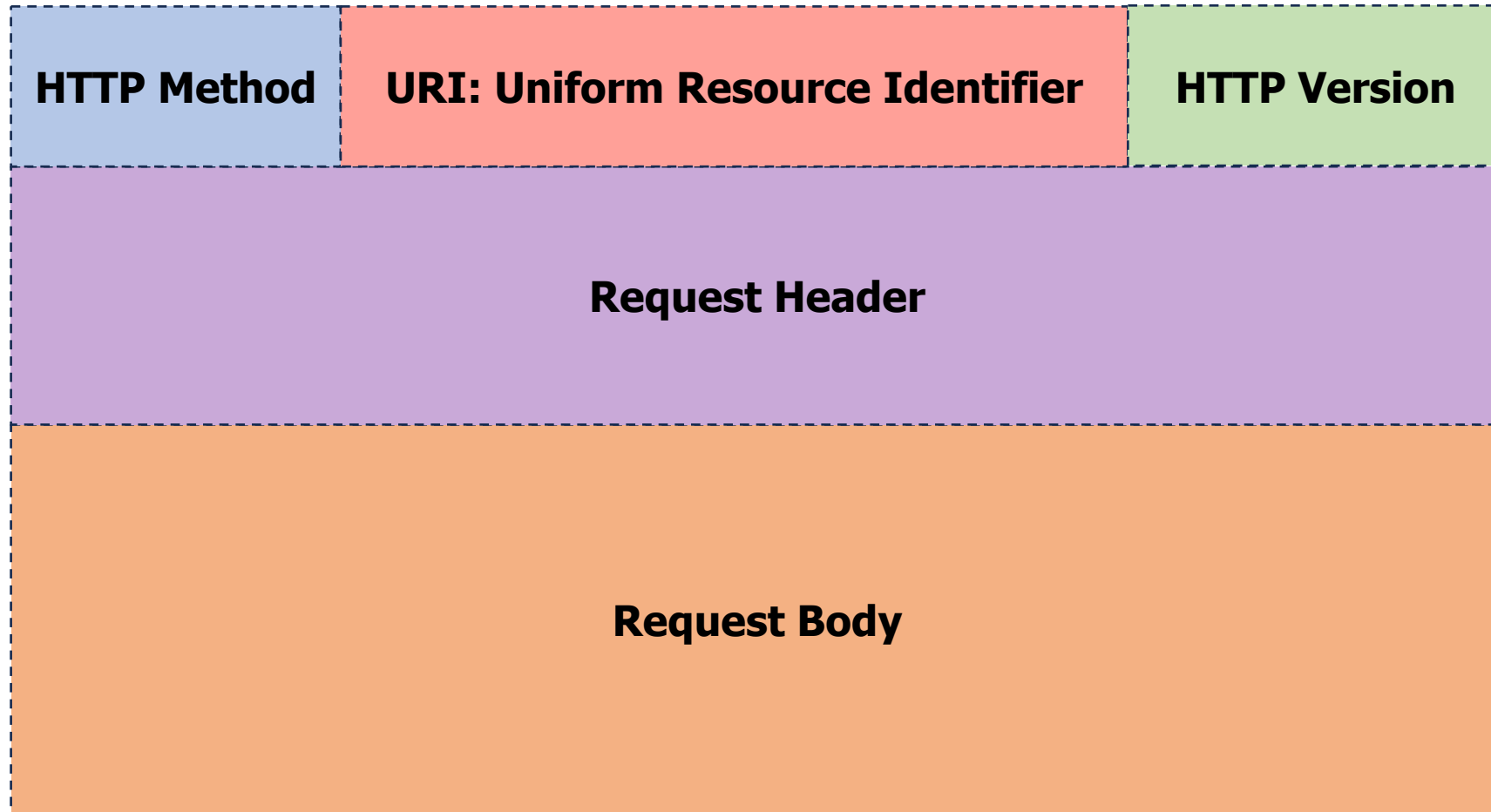
C<sub>reate</sub> → POST

R<sub>ead</sub> → GET

U<sub>pdate</sub> → PUT

D<sub>ele</sub>te → DELETE

# HTTP Request



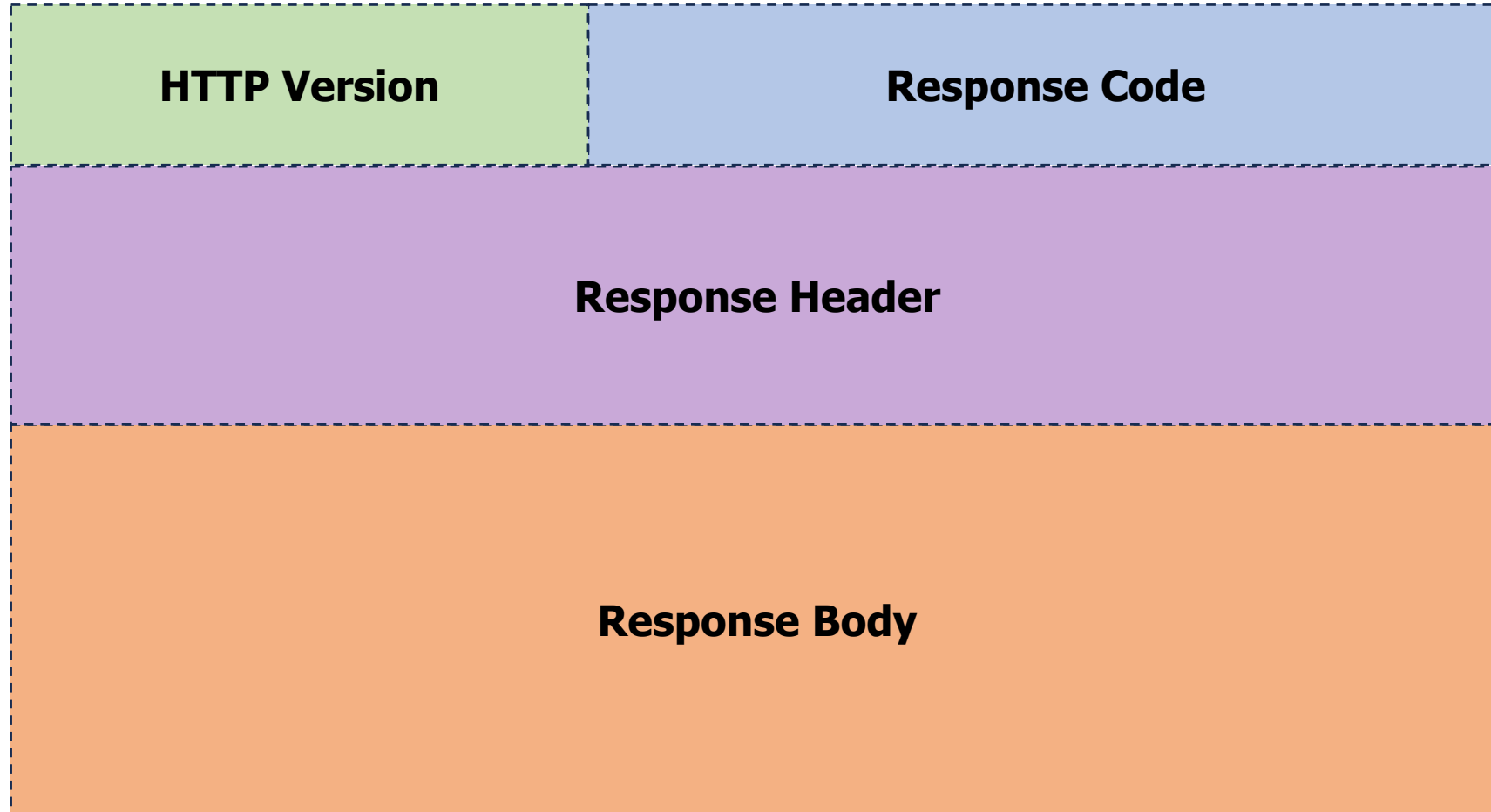


# GET Request

## `http://urpizza.com/api/toppings`

GET	<code>http://urpizza.com/api/toppings</code>	HTTP/1.1
<code>Accept: application/json</code> <code>User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X)</code>		

# HTTP Response



# GET Response

## <http://urpizza.com/api/toppings>

HTTP/1.1

200 OK

Content-Length: 32859

Content-Type: application/json

```
[{"id": "1", "topping": "mozzarella"},  
{"id": "2", "topping": "green pepper"},  
{"id": "3", "topping": "black olive"},  
{"id": "4", "topping": "red onion"},  
{"id": "5", "topping": "mushroom"},  
{"id": "6", "topping": "pepperoni"}]
```

# JSON (Java Script Object Notation)

- JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "string": "Hello World",
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d"
  }
}
```

# HTTP Response Codes For Success

- 2xx success
  - 200 OK
    - Standard response for successful HTTP requests
    - Use for successful GET and PUT requests
  - 201 Created
    - The request has been fulfilled, resulting in the creation of a new resource
    - Use for successful POST requests
  - 204 No Content
    - The server successfully processed the request, and is not returning any content
    - Use for successful DELETE requests

# HTTP Response Codes For Client Errors

- 4xx client errors
  - 400 Bad Request
    - The server cannot or will not process the request due to an apparent client error e.g., malformed request syntax, size too large, invalid request message
    - Use for unsuccessful POST and PUT requests when JSON parsing fails
  - 401 Unauthorized
    - When authentication is required and has failed or has not yet been provided
  - 404 Not Found
    - The requested resource could not be found
    - Use for unsuccessful GET, PUT, and DELETE requests when resource is not found
  - 418 I'm a teapot
    - The server refuses to brew coffee because it is, permanently, a teapot
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>





# GET Request

## `http://urpizza.com/api/toppings`

GET	<code>http://urpizza.com/api/toppings/2</code>	HTTP/1.1
<code>Accept: application/json</code> <code>User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X)</code>		

# GET Response

## <http://urpizza.com/api/toppings>

HTTP/1.1	200 OK
Content-Length: 859 Content-Type: application/json	
{ "id": "2", "topping": "green pepper" }	

# PUT Request

## `http://urpizza.com/api/toppings`

<code>PUT</code>	<code>http://urpizza.com/api/toppings/2</code>	<code>HTTP/1.1</code>
<code>Accept: application/json</code> <code>User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X)</code>		
<code>{"id": "2", "topping": "red pepper"}</code>		

# PUT Response

## <http://urpizza.com/api/toppings>

HTTP/1.1	200 OK
Content-Length: 859 Content-Type: application/json	
{ "id": "2", "topping": "red pepper" }	

# POST Request

## http://urpizza.com/api/toppings

POST	http://urpizza.com/api/toppings	HTTP/1.1
Accept: application/json User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X)		
{ "id": "7", "topping": "pineapple" }		

# POST Response

## <http://urpizza.com/api/toppings>

HTTP/1.1	201 Created
Content-Length: 859 Content-Type: application/json	
{ "id": "7", "topping": "pineapple" }	



# C++ Library for RESTful web service



A Fast and Easy to use microframework for the web.

Crow is a C++ framework for creating HTTP or Websocket web services. It uses routing similar to Python's Flask which makes it easy to use. It is also extremely fast, beating multiple existing C++ frameworks as well as non C++ frameworks.



Blazingly Fast



Header Only




Typesafe handlers



Websocket Support

<https://crowcpp.org>

# Hello REST



```
#include <crow.h>
#include <string>
using namespace std;
using namespace crow;
```



```
response handleHelloGetRequest()
```

```
{
    string body = "Hello, world!";
```

```
// Return the integer HTTP response code, and the response body as a string.
```



```
return response(200, body);
}
```

```
int main()
{
```

```
// Create a simple crow application.
```



```
SimpleApp app;
```

```
// Define your endpoint and provide a function to handle the request.
```



```
CROW_ROUTE(app, "/api/hello").methods(HTTPMethod::Get)(handleHelloGetRequest);
```

```
// Set the port and run the app.
```



```
app.port(18080).run();
}
```