

Quickest Path API - User Manual

Introduction.....	1
Algorithm Overview.....	1
Getting Started.....	3
How to Download the API?.....	3
Start the API Server.....	3
If the command was successful, the terminal should display:.....	3
How to Construct a Correct Request?.....	4
API Usage Examples.....	6
Using a Web Browser.....	6
Using curl (Command Line).....	6
Using Python (requests Library).....	7
Using Postman (API Testing Tool).....	8
Support.....	8
Appendix.....	10

Introduction

Welcome to the Quickest Path API documentation. This guide provides an overview of how to use the API to retrieve the fastest route between two landmarks using simple HTTP requests. It includes example queries, response formats, and error handling.

This project being open-source, feel free to contribute or look at the code on our GitHub repository:

<https://github.com/algosup/2024-2025-project-3-quickest-path-team-8>

Algorithm Overview

The Quickest Path API is powered by a dataset containing **23,947,347 unique landmarks** across the United States. These landmarks are interconnected via weighted paths but do not have associated names—only numerical IDs.

To compute the shortest route between two landmarks, the API employs a **bidirectional Dijkstra algorithm**, ensuring efficient pathfinding with an average response time of **less than 2 seconds**.


Getting Started

How to Download the API?

Download the API by cloning our repository:

```
git clone  
https://github.com/algosup/2024-2025-project-3-quickest-path-team-8
```

Alternatively, you can manually download the project:

1. Open a web browser and go to the GitHub repository.
2. Click on , then select **Download ZIP**.
3. Once the file is downloaded, go to your **Downloads** folder and double click on the ZIP file.

Start the API Server

Verify you have g++ installed on your machine.

1. After downloading, open a terminal and move to the `src` folder of the extracted project directory:

```
cd path/to/the/project/src
```

2. To start the API server, run the following command in your terminal

```
cd src && g++ -std=c++17 -O1 -march=native main.cpp graph.cpp dijkstra.cpp  
binary.cpp rest_api.cpp -o api && ./api
```

If the command was successful, the terminal should display:

```
Live server started on localhost: http://127.0.0.1:8080
```

If you see an error message or nothing happens:

- Ensure all dependencies are correctly installed.
- Try running step 2 again.
- If the issue persists, create a bug report on the **Issues** section of our GitHub

repository.

How to Construct a Correct Request?

Base URL:

`http://127.0.0.1:8080`

Endpoint:

`GET /quickest_path`

Parameters:

Parameter	Type	Description
<code>landmark_1</code>	Integer	The starting landmark ID
<code>landmark_2</code>	Integer	The destination landmark ID
<code>format</code>	String	Response format: <code>json</code> or <code>xml</code>

To ensure a valid API request, follow these steps:

1. Identify the required landmarks:
 - Choose two landmark IDs within the dataset range (from 1 to 23,947,347).
 - Example: Start at `1` and end at `2`.
2. Select a valid response format:
 - Supported formats: `json` or `xml`.
 - Example: Choose `json` for a structured response.
3. Structure the request URL correctly:
 - Base URL: `http://127.0.0.1:8080`
 - Append the endpoint: `/quickest_path`
 - Include query parameters in the correct format:
`?format=json&landmark_1=1&landmark_2=2`

Example Request:

`http://127.0.0.1:8080/quickest_path?format=json&landmark_1=1&landmark_2=2`

In this example:

- The first landmark (`landmark_1`) is `1`.
- The second landmark (`landmark_2`) is `2`.
- The response format is `json`.

By following these steps, you can properly structure your requests and retrieve the desired path information.

Expected Response (JSON):

```
{
  "distance": 10,
  "path": [
    1,
    2
  ]
}
```

Expected Response (XML):

```
<response>
  <distance>10</distance>
  <path>
    <landmark>1</landmark>
    <landmark>2</landmark>
  </path>
</response>
```

Example Invalid Request:

```
http://127.0.0.1:8080/quickest_path?format=html&landmark_1=ABC&landmark_2=9999999999
```

In this example:

- The first landmark (`landmark_1`) is invalid (`ABC` instead of an integer).
- The second landmark (`landmark_2`) is out of range (`9999999999`).
- The response format is incorrect (`html` instead of `json` or `xml`).

Expected Error Response:

When multiple errors are detected, only the first one is returned—for instance, the invalid return format error in the example below.

```
{
  "error": {
    "code": 401,
    "message": "Invalid Format",
    "details": "Supported formats: json, xml"
  }
}
```

By following these steps, users can properly structure their requests and retrieve the desired path information, if you encounter an error, check the [appendix](#) at the end of the document.

API Usage Examples

Using a Web Browser

Simply enter the API request URL in the address bar to view the response.

- If the format is JSON, browsers like Chrome or Firefox may require a JSON viewer extension for better readability.
- If the format is XML, browsers will natively render the structured response.

Using `curl` (Command Line)

`curl` is a command-line tool for making HTTP requests.

Basic Example:

```
curl -X GET  
"http://127.0.0.1:8080/quickest_path?format=json&landmark_1=1&landmark_2=2"
```

Saving the Response to a File:

```
curl -X GET  
"http://127.0.0.1:8080/quickest_path?format=json&landmark_1=1&landmark_2=2"  
-o response.json
```

Using Python ([requests](#) Library)

Python provides a convenient way to interact with the API programmatically.

Basic Example:

```
import requests  
  
url = "http://127.0.0.1:8080/quickest_path"  
params = { "format": "json", "landmark_1": 1, "landmark_2": 2 }  
response = requests.get(url, params=params)  
print(response.json())
```

Handling Errors Gracefully:

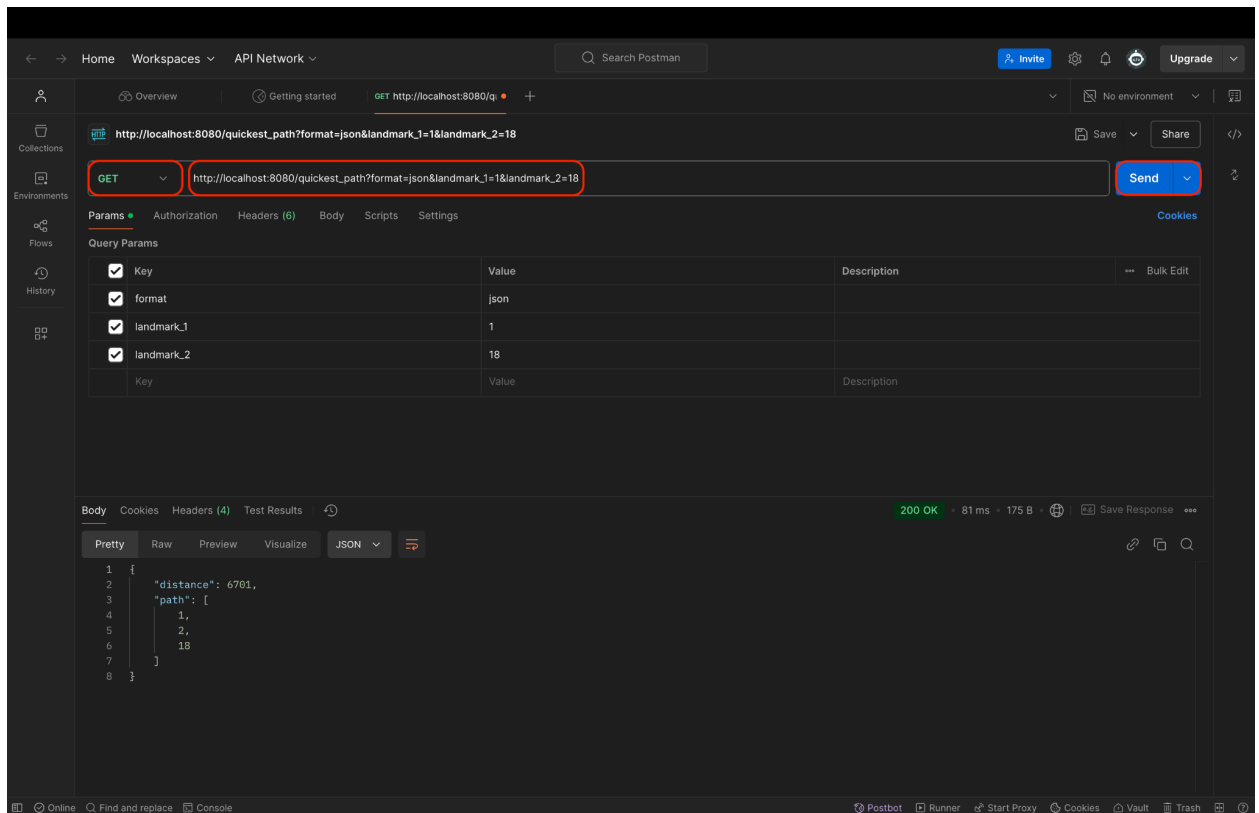
```
try:  
    response = requests.get(url, params=params)  
    response.raise_for_status()  
    data = response.json()  
    print("Shortest path:", data)  
except requests.exceptions.RequestException as e:  
    print("Error:", e)
```

Using Postman (API Testing Tool)

1. Open [Postman](#).
2. Select **GET** as the request method.
3. Enter the API endpoint:

http://127.0.0.1:8080/quickest_path?format=json&landmark_1=1&landmark_2=18

4. Click **Send**.
5. View the formatted response in the [Postman](#) response panel.



Support

Our API incorporates comprehensive error handling to help users troubleshoot their requests. Each error response includes an HTTP status code and a descriptive error message. For a complete overview of these errors, please refer to the [appendix](#) at the end of this document.

If you need assistance with API requests or encounter any issues, we offer multiple ways to get support:

- **GitHub Issues:** Report bugs or feature requests on our GitHub repository: [Quickest Path API Issues](#)
- **Quality Assurance Contact:** For urgent issues, contact our QA representative by email: max.bernard@algosup.com

Our team will respond as soon as possible to ensure a smooth experience with the API.

Appendix

Error Scenario	HTTP Code	Description	Example JSON Response	Example XML Response	Solution
Missing parameters	400	One or more required parameters (landmark_1 , landmark_2 , format) are missing.	<pre>{ "error": { "code": 400, "message": "Missing Parameters", "details": "Required parameters: format, landmark_1, landmark_2" } }</pre>	<pre><error> <code>400</code> <message>Missing Parameters</message> <details>Required parameters: format, landmark_1, landmark_2</details> </error></pre>	Ensure landmark_1 and landmark_2 and format are included in the request.
Invalid Format	401	The specified return format is not supported. Only json and xml are allowed.	<pre>{ "error": { "code": 401, "message": "Invalid Format", "details": "Supported formats: json, xml" } }</pre>	<pre><error> <code>401</code> <message>Invalid Format</message> <details>Supported formats: json, xml</details> </error></pre>	Check if the landmark_1 and landmark_2 exist in the dataset before making a request.
Invalid Landmark Type	402	The data type of one or more landmark IDs is invalid. Landmark IDs must be integers.	<pre>{ "error": { "code": 402, "message": "Invalid Landmark Type", "details": "Landmark IDs must be valid integers" } }</pre>	<pre><error> <code>402</code> <message>Invalid Landmark Type</message> <details>Landmark IDs must be valid integers</details> </error></pre>	Verify that landmark_1 and landmark_2 are defined as integers in your request.
Invalid Landmark Range	404	One or more landmark IDs are outside the valid range (1 to 23947347).	<pre>{ "error": { "code": 404, "message": "Invalid Landmark Range", "details": "Landmark IDs must be between 1 and 23947347" } }</pre>	<pre><error> <code>404</code> <message>Invalid Landmark Range</message> <details>Landmark IDs must be between 1 and 23947347</details> </error></pre>	Ensure that the values for landmark_1 and landmark_2 fall within the valid range (1 to 23947347).
Internal Server Error	500	An unhandled error occurred during request processing.	<pre>{ "error": { "code": 500, "message": "Internal Server Error", "details": "An unexpected error occurred" } }</pre>	<pre><error> <code>500</code> <message>Internal Server Error</message> <details>An unexpected error occurred</details> </error></pre>	Retry after some time. If the issue persists, contact support.