Project Description

The URL Shortener project is a web service that provides the ability to create shortened URLs and redirect to the original long URLs. The service includes two main API endpoints:

POST /shorten: Takes a JSON request with a long URL and returns a shortened URL.

GET /{shortURL}: Redirects to the original long URL.

Project structure

```
url-shortener/
├── main.go
├── main_test.go
├── go.mod
└── go.sum
```

Main Components

main.go: main file with application logic.

main_test.go: file with tests to check the correct operation of the service.

go.mod and go.sum: files that manage Go dependencies.

Installation and launch

Requirements

Go 1.18 or higher

MariaDB/MySQL server

Database setup

Create a urlshortener database and a urls table:

```
CREATE DATABASE urlshortener;

USE urlshortener;

CREATE TABLE urls (
```

```
 id INT AUTO_INCREMENT PRIMARY KEY,

original_url TEXT NOT NULL,
short_url VARCHAR(255) NOT NULL UNIQUE
 );
 short_url VARCHAR(255) NOT NULL UNIQUE
 );
```

Setting up and launching the application

Clone the repository

```
git clone https://github.com/yourusername/url-shortener.git
cd url-shortener
```

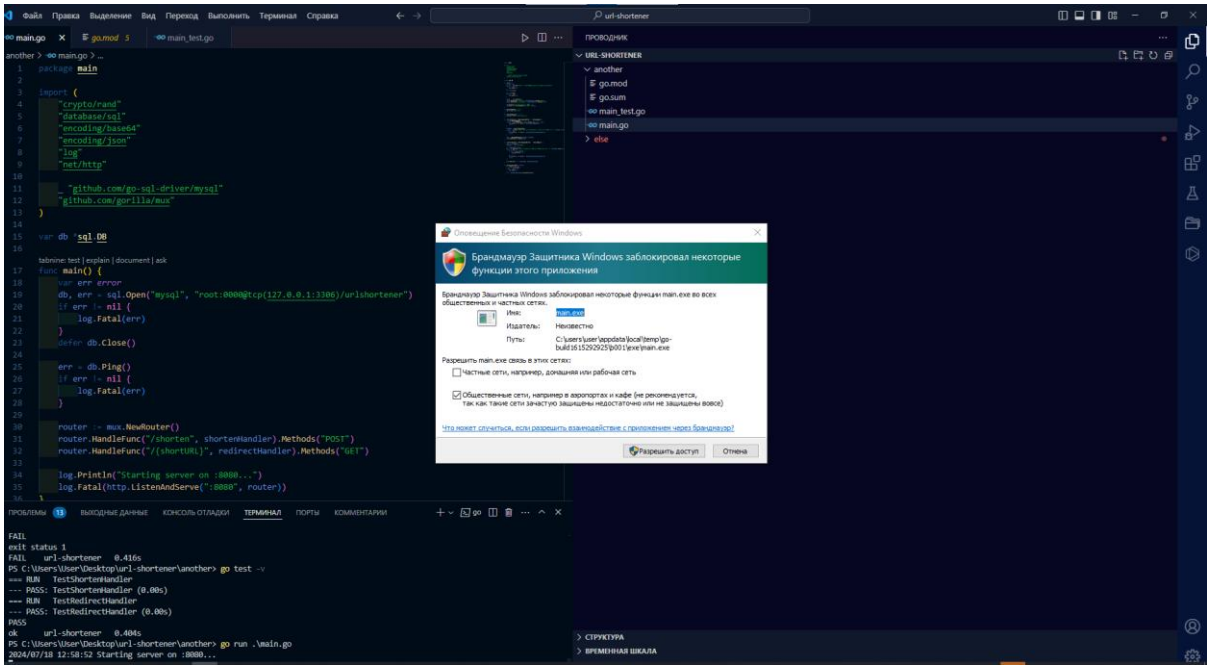
Install dependencies:

```
go mod tidy
```

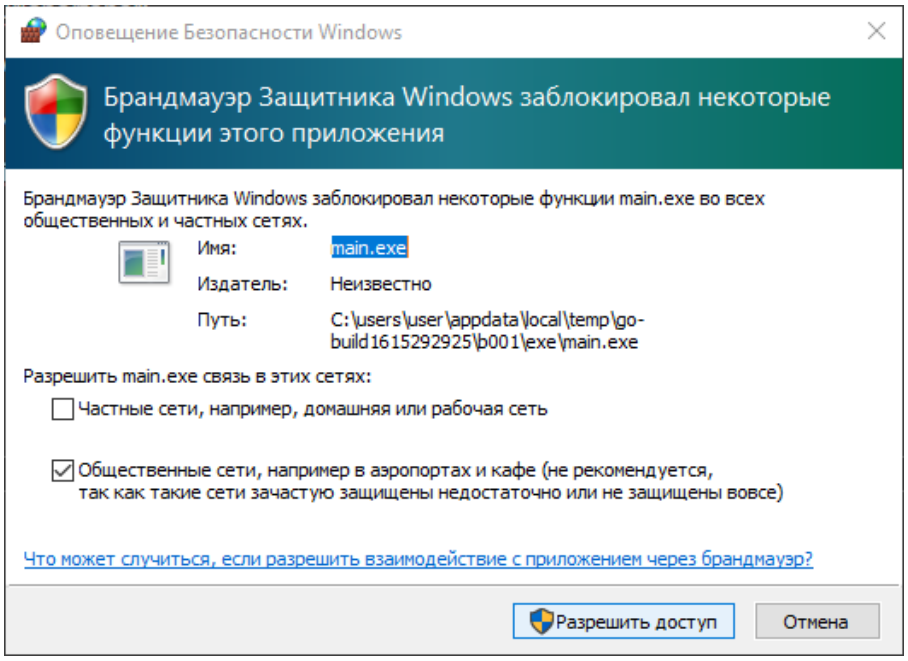
Launch the application:

```
go run main.go
```

Run main.go using the command go run .\main.go



Let's start the process



Workflow display

```
2024/07/18 12:58:52 Starting server on :8080...
```

API endpoints

POST /shorten

Description: Takes a JSON request with a long URL and returns a shortened URL.

Example request:

```
{
"url": "http://example.com"
}
```

Sample answer:

```
{
 "short_url": "randomShortURL"
}
```

Postman example

POST /shorten

In Postman, create a new POST request.

Set the URL to http://localhost:8080/shorten.

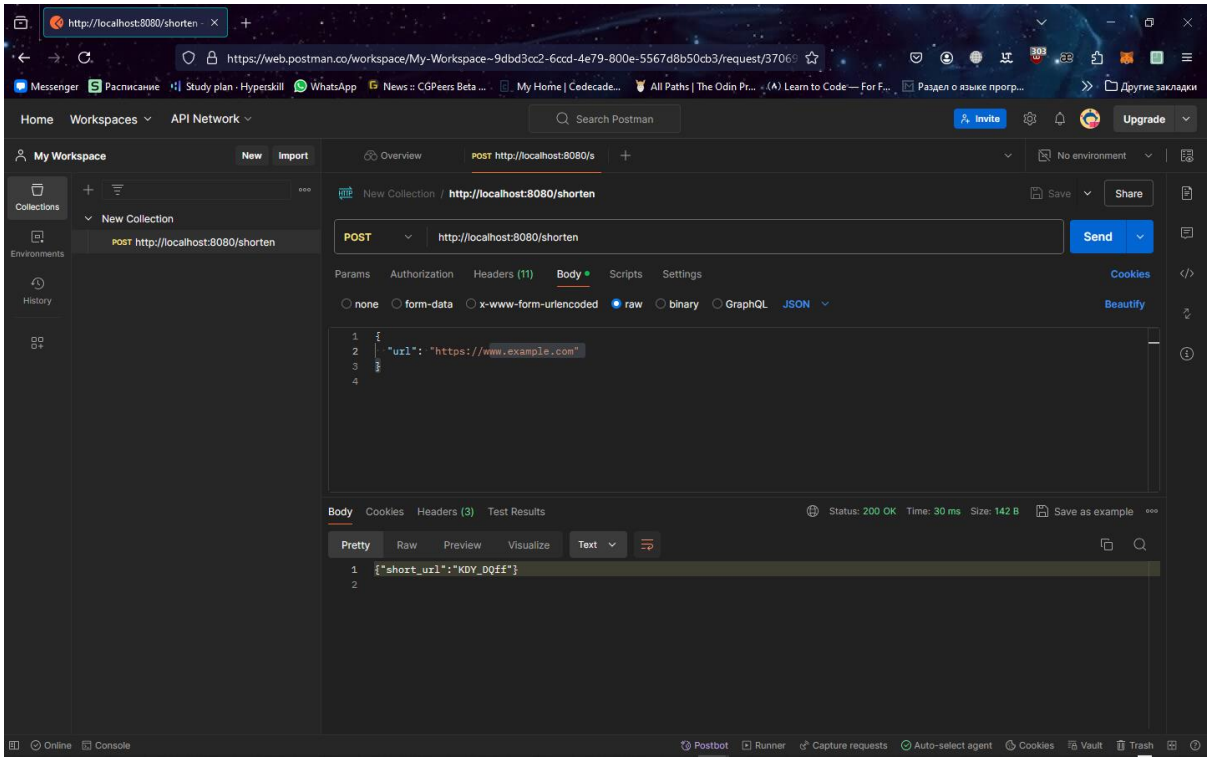
Go to the Body tab, select raw and set the format to JSON.

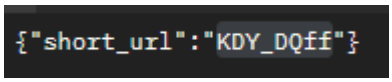
Enter JSON:

We make a POST request and receive KDY_DQff randomly generated using the generateShortURL function

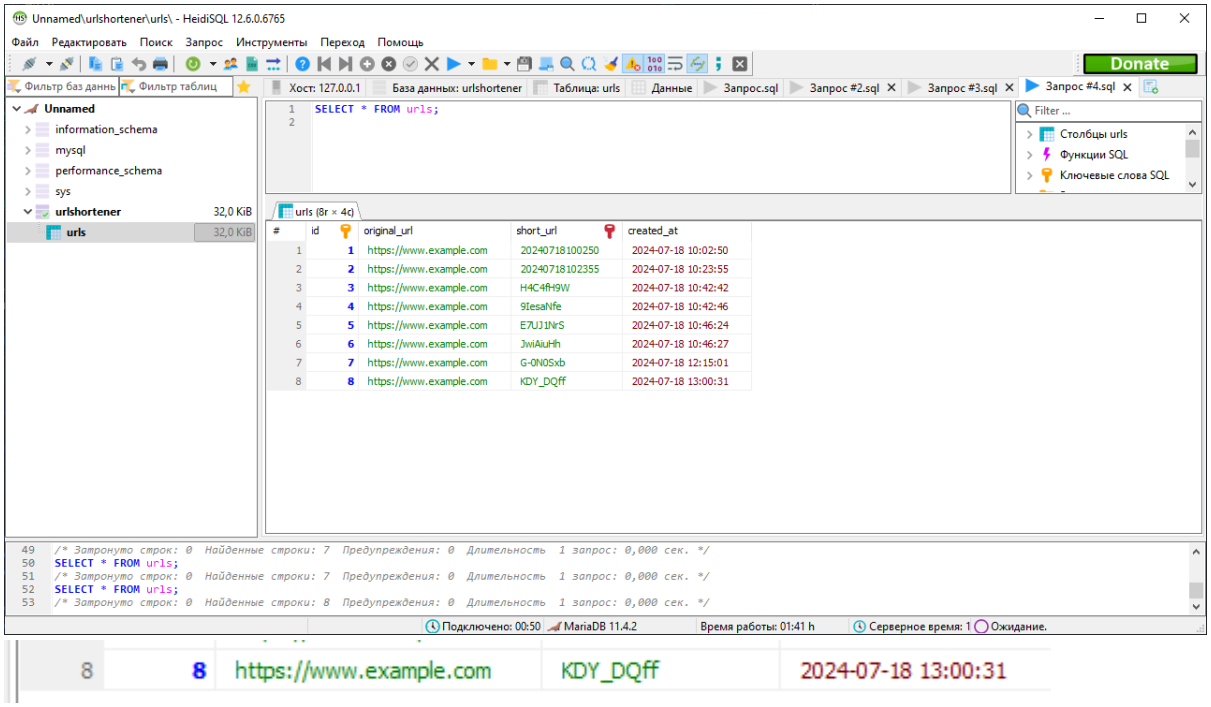
```
 {
 "url": "http://example.com"
 }
```

Click Send. You will receive a shortened URL in the response.



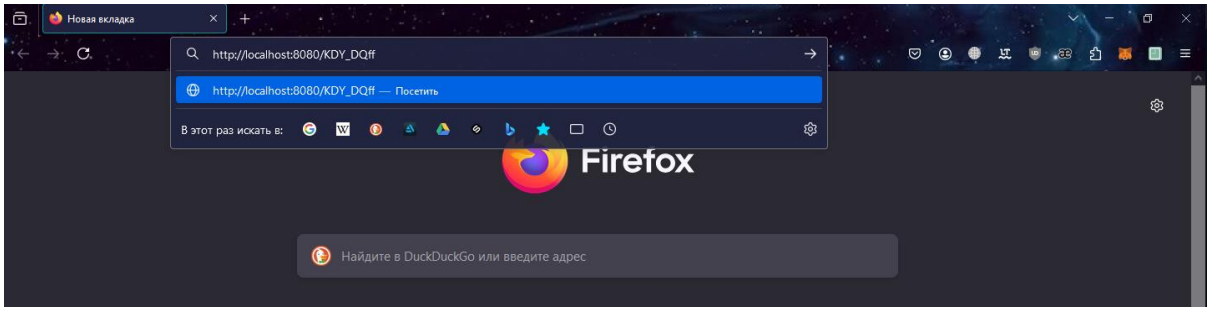


We update the database and see the display in it

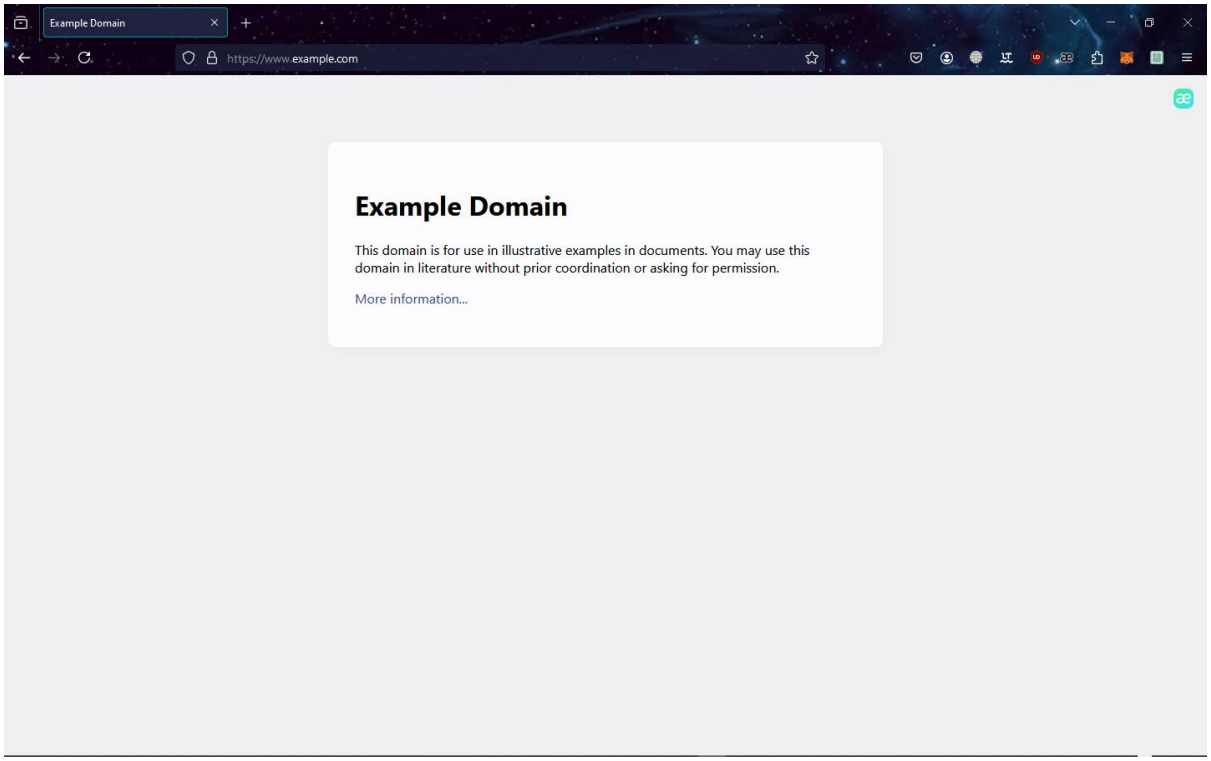


Go to a locally running server at address 8080

http://localhost:8080/shorten/KDY_DQff

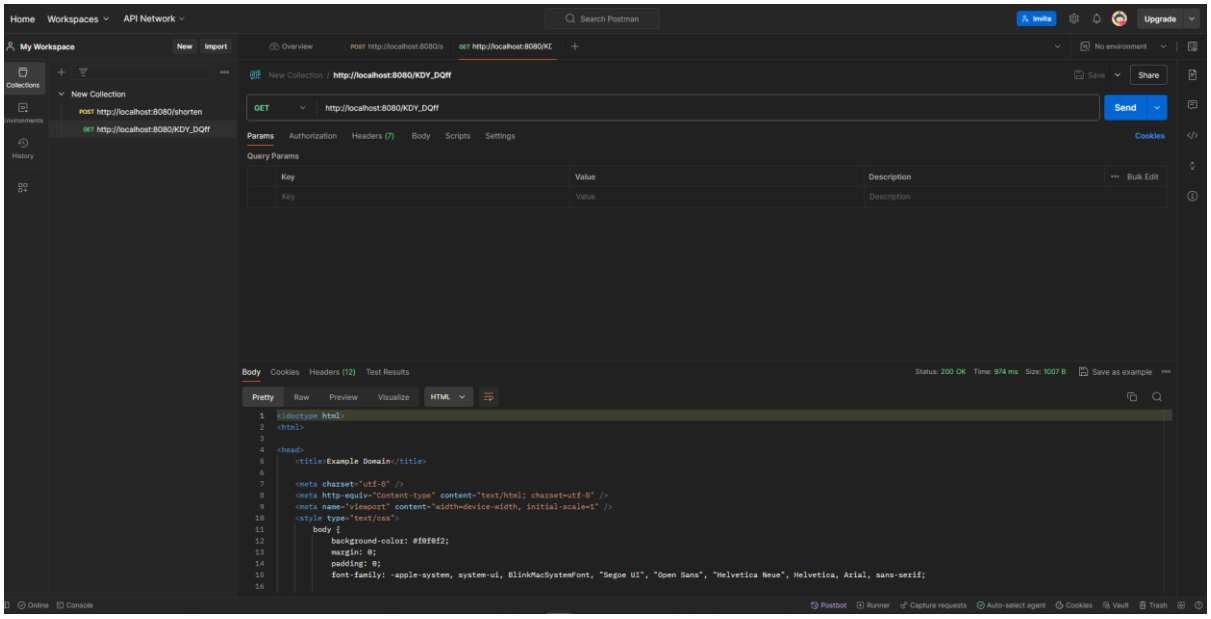


We receive in response a page about using the exemple stub



GET /{shortURL}

In Postman, create a new GET request.
Set the URL to http://localhost:8080/{shortURL}, where {shortURL} is replaced with the shortened URL obtained earlier for example KDY_DQff
Click Send. You will be redirected to the original long URL



Testing

To run tests use the command:

go test -v

Performing tests

```
PS C:\Users\User\Desktop\url-shortener\another> go test -v
=== RUN   TestShortenHandler
--- PASS: TestShortenHandler (0.00s)
=== RUN   TestRedirectHandler
--- PASS: TestRedirectHandler (0.00s)
PASS
ok      url-shortener   0.404s
```

Testing the TestShortenHandler function.

This function handles a POST request to the /shorten endpoint, which takes a JSON payload with a URL and returns a shortened version of that URL.

The test uses the sqlmock package to create a mock database and sets expectations for database operations.

In this case, the INSERT statement is expected to be executed with the specified URL and the generated short URL.

The test creates a POST request with a JSON payload containing the URL to shorten. It then configures a response record to capture the HTTP response. The http.HandlerFunc(shortenHandler) function is called to process the request and write the response to rr.

After this, checks are performed using the assert package: the response code is expected to be http.StatusOK (200) and the returned short URL must not be empty. Finally, the test checks if all expected expectations have been met using mock.ExpectationsWereMet().

If there are unfulfilled waits, the test fails.

Test function TestRedirectHandler for the HTTP request handler redirectHandler.

This feature is responsible for redirecting users to the original URL based on the short URL specified in the request path.

The test uses the sqlmock package to create a mock database and expectations for the SQL queries that will be executed during the test.

The mockDB object is used to replace the actual database connection in the application code. When making a GET request to the /shortURL123 route, the redirectHandler is expected to return a response with a status code of 302 (Found) and a Location header with the value "http://example.com", which points to the original URL.

After executing the request and receiving the response, the test checks if all expected expectations were met using mock.ExpectationsWereMet().

If there are unfulfilled expectations, the test throws an error.