A BDD Approach to Testing a URL Shortener in Go

https://github.com/ItsDobiel/URLShortener

Kiarash Ghorbani

4041 Semester @ SRBIAU

URL Shortener Architecture

How Short Codes Are Generated

The Algorithm

Our URL shortener uses a **hash-based approach** to generate short codes:

1. **Input**: Original URL (e.g., `https://example.com/very/long/path`)
2. **Hash Function**: SHA-256 cryptographic hash
3. **Encoding**: Base64 URL-safe encoding
4. **Truncation**: Take first 7 characters

```
// Pseudocode
hash = SHA256(normalized_url + ":" + attempt_number)
encoded = Base64URLEncode(hash)
shortCode = encoded[0:7]  // e.g., "dmOa-QD"
```

Why SHA-256 and Base64?

SHA-256 Hash Function

- **Deterministic**: Same input → same output
- **Uniform distribution**: Minimizes collisions
- **One-way**: Cannot reverse to get original URL
- **Fast**: Efficient computation

Base64 URL-Safe Encoding

- **Compact**: 64 characters (A-Z, a-z, 0-9, -, _)
- **URL-safe**: No special characters that need escaping
- **Efficient**: 6 bits per character
- **7 characters**: ~64^7 = 4 trillion possible combinations

Note that in BDD Testing we don’t need this infomation, It is only provided as extra knowledge.

URL Normalization

Why Normalize URLs?

The Problem

Users might enter the same URL in different ways:

- `HTTPS://GITHUB.COM/mozilla/`
- `https://github.com/mozilla`
- `https://GitHub.com/mozilla/`

Without normalization → **different short codes** for the **same resource**!

Normalization Examples

Before and After

Original URL	Normalized URL
`HTTPS://GITHUB.COM/mozilla/`	`https://github.com/mozilla`
`https://SRB.IAU.ir/library/fa/page/6151/`	`https://srb.iau.ir/library/fa/page/6151`

Result: All variations → **same short code**

Key: Protocol and domain are case-insensitive, path is case-sensitive

Collision Handling

What is a Hash Collision?

The Problem

Two different URLs → **same short code**

```
URL1: "https://github.com/ItsDobiel/URLShortener" → Hash → "abc1234"
URL2: "https://srb.iau.ir/library/fa/page/6151"   → Hash → "abc1234" ❌
```

While rare with SHA-256, collisions are still possible! Also keep in mind We can't test this since We don't have urls that produce the same SHA-256 hash!

Behavior-Driven Development (BDD)

What is BDD?

Definition

Behavior-Driven Development is a software development approach that:

- Uses **plain English** to describe system behavior
- Focuses on **user stories** and scenarios
- Enables **collaboration** between developers, testers, and stakeholders
- Serves as **living documentation**

Key Principle

> *“Tests should describe **what** the system does, not **how** it does it”*

Gherkin Syntax

The Language of BDD

Gherkin is a plain-text language with keywords:

- `Feature`: High-level description
- `Background`: Steps to be executed within each Scenario
- `Scenario Outline`: Specific test case
- `Given`: Initial context/state
- `When`: Action/event
- `Then`: Expected outcome
- `And`: Additional conditions

Gherkin Example

Basic Structure

```
Feature: URL Shortener
  As a user
  I want to shorten long URLs
  So that I can share them easily

  Background:
    Given the URL shortener service is running
    And I am on the home page

  Scenario Outline: Shorten valid URLs
    When I enter the URL "https://github.com/ItsDobiel/URLShortener/blob/main/README.md"
    And I submit the form
    Then I should see a success message
    And I should see a shortened URL
    And the shortened URL should be valid
    And the shortned URL must redirect me to "https://github.com/ItsDobiel/URLShortener/blob/main/README.md"
```

Scenario Outline with Examples

Data-Driven Testing

We will update the previous example!

```
Scenario Outline: Shorten valid URLs
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a success message
  And I should see a shortened URL
  And the shortened URL should be valid
  And the shortned URL must redirect me to "<url>"

  Examples:
    | url                                                          |
    | https://github.com/ItsDobiel/URLShortener/blob/main/README.md |
    | https://cucumber.io/docs/bdd/                                |
```

Result: Test runs **twice** with different data

Our Test Scenarios

Based on URL Shortener Features

1. Shortened URL Testing

```
Scenario Outline: Shorten valid URLs
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a success message
  And I should see a shortened URL
  And the shortened URL should be valid
  And the shortned URL must redirect me to "<url>"

  Examples:
    | url                                                     |
    | https://github.com/ItsDobiel/URLShortener/blob/main/README.md |
    | https://cucumber.io/docs/bdd/                           |
```

Tests: Tests the generation of a functional shortened url and opens it to verify redirection.

2. Normalization Testing

```
Scenario Outline: URL normalization - trailing slashes
  When I enter the URL "<url_with_slash>"
  And I submit the form
  Then I should see a shortened URL
  When I enter the URL "<url_without_slash>"
  And I submit the form
  Then I should receive the same short code as before

  Examples:
    | url_with_slash                            | url_without_slash                        |
    | https://gorm.io/docs/                     | https://gorm.io/docs                     |
    | https://github.com/ItsDobiel/URLShortener/ | https://github.com/ItsDobiel/URLShortener |

Scenario Outline: URL normalization - case insensitivity in protocol and domain
  When I enter the URL "<url_variant1>"
  And I submit the form
  Then I should see a shortened URL
  When I enter the URL "<url_variant2>"
  And I submit the form
  Then I should receive the same short code as before

  Examples:
    | url_variant1                              | url_variant2                              |
    | HTTPS://GiThUb.CoM/ItsDobiel/URLShortener | https://github.com/ItsDobiel/URLShortener |
    | httpS://docs.PODMAN.io/en/latest          | https://docs.podman.io/en/latest          |
```

Tests: Protocol/domain normalization, duplicate detection

3. Duplicate Handling Testing

```
Scenario Outline: Duplicate URL returns same short code
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a shortened URL
  When I enter the URL "<url>"
  And I submit the form
  Then I should receive the same short code as before

  Examples:
    | url                                              |
    | https://www.gnu.org/software/bash/manual/bash.html |
    | https://www.mozilla.org/en-US/about/manifesto      |
```

Tests: Database lookup, collision avoidance

4. Handling Query Parameters, Fragments and Special Characters

```
Scenario Outline: Shorten URLs with query parameters, fragments and special characters
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a success message
  And I should see a shortened URL

  Examples:
    | url                                                              |
    | https://example.com/search?q=Price%20of%20US%20%24&page=2#results  |
    | https://example.com/post?uuid=c3191902-bbef-4434-b043-e2cceedcc227 |
```

Tests: Shortened url generation

5. Validation Testing

```
Scenario Outline: Reject invalid URLs
  When I enter the URL "<invalid_url>"
  And I submit the form
  Then I should see an error message

  Examples:
    | invalid_url              |
    | not-a-valid-url          |
    | ftp://example.com/file   |
    | javascript:alert('xss')  |
    | file:///etc/passwd       |
    | /../../../etc/passwd     |
    | /invalid@chars!          |
    | /short code with spaces  |
```

Tests: Input validation, security

6. Undefined Short Code Testing

```
Scenario Outline: Accessing non-existent short code
  When I navigate to "<path>"
  Then I should see an error page
  And the error should indicate the short code was not found

  Examples:
    | path        |
    | /AuEcAowlXq |
    | /XXXXXXX    |
```

Tests: Handling of undefined short codes

5. Short Code Format Validation

```
Scenario Outline: Generated short code meets requirements
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a shortened URL
  And the short code should be alphanumeric with allowed characters
  And the short code length should match the configured length

  Examples:
    | url                                          |
    | https://github.com/ItsDobiel/URLShortener       |
    | https://docs.fedoraproject.org/en-US/containers |
```

Tests: Base64 encoding, configured length (7 chars)

Testing in Go with Godog

What is Godog?

BDD Framework for Go

Godog is the Go implementation of Cucumber:

- Reads `.feature` files written in Gherkin
- Maps steps to Go functions
- Integrates with Go's `*testing.T`
- Provides detailed test reports

```go
import "github.com/cucumber/godog"
```

Project Structure

Test Organization

```
test/
├── features/
│   └── url_shortener.feature    # Gherkin scenarios
└── url_shortener_test.go        # Go test code
```

Separation of concerns: - Feature files → **What** to test (business logic) - Go code → **How** to test (implementation)

Step 1: Define Feature File

Example Scenario

```gherkin
Scenario Outline: Shorten valid URLs
  When I enter the URL "<url>"
  And I submit the form
  Then I should see a success message
  And I should see a shortened URL
  And the shortened URL should be valid
  And the shortned URL must redirect me to "<url>"

  Examples:
    | url                                                          |
    | https://github.com/ItsDobiel/URLShortener/blob/main/README.md |
    | https://cucumber.io/docs/bdd/                                |
```

Step 2: Create Test Entry Point

Main Test Function

```go
func TestFeatures(t *testing.T) {
    // Configure godog test suite
    suite := godog.TestSuite{
        ScenarioInitializer: initializeScenario,
        Options: &godog.Options{
            Format:   "pretty",
            Paths:    []string{"features"},
            TestingT: t,  // Integration with Go testing
        },
    }

    // Run the suite
    if suite.Run() != 0 {
        t.Fatal("tests failed")
    }
}
```

Step 3: Register Step Definitions

Mapping Gherkin to Go

```go
func initializeScenario(ctx *godog.ScenarioContext) {
    // Register step definitions
    ctx.Step(`^I enter the URL "([^"]*)"$`, stepEnterURL)
    ctx.Step(`^I submit the form$`, stepSubmitForm)
    ctx.Step(`^I should see a success message$`, stepSeeSuccessMessage)
    ctx.Step(`^I should see a shortened URL$`, stepSeeShortenedURL)
    ctx.Step(`^the shortened URL should be valid$`, stepShortenedURLValid)
    ctx.Step(`^the shortned URL must redirect me to "([^"]*)"$`, stepNavigateToShortenedURL)
}
```

Note: Regular expressions capture parameters

Step 4: Implement Step Functions

Example: “I enter the URL”

```go
func stepEnterURL(url string) error {
    fmt.Printf("   Entering URL: %s\n", url)

    currentURL, _ := testCtx.webDriver.CurrentURL()
    if !strings.HasSuffix(currentURL, "/") || strings.Contains(currentURL, "error") {
        fmt.Println("   Navigating back to home page...")
        testCtx.webDriver.Get(testCtx.baseURL)
    }

    urlInput, err := testCtx.webDriver.FindElement(selenium.ByID, "url")
    if err != nil {
        return fmt.Errorf("URL input not found: %w", err)
    }

    urlInput.Clear()
    err = urlInput.SendKeys(url)

    return err
}
```

Parameter: `url` is captured from Gherkin step

Complete Example: Reject Invalid URLs

Feature File

```gherkin
Scenario Outline: Reject invalid URLs
  When I enter the URL "<invalid_url>"
  And I submit the form
  Then I should see an error message

  Examples:
    | invalid_url              |
    | not-a-valid-url          |
    | ftp://example.com/file   |
    | javascript:alert('xss')  |
    | file:///etc/passwd       |
    | /../../../etc/passwd     |
    | /invalid@chars!          |
    | /short code with spaces  |
```

Step Implementations

The error handling code is redacted for simplicity.

```go
func stepEnterURL(url string) error {
    urlInput, err := testCtx.webDriver.FindElement(selenium.ByID, "url")
    urlInput.Clear()
    return err
}

func stepSubmitForm() error {
    button, _ := testCtx.webDriver.FindElement(selenium.ByID, "submit")
    button.Click()
    return nil
}

func stepSeeSuccessMessage() error {
    pageSource, _ := testCtx.webDriver.PageSource()
    if strings.Contains(pageSource, "Success") || strings.Contains(pageSource, "✨") {
        return nil
    }

    elements, _ := testCtx.webDriver.FindElements(selenium.ByID, "result")
    if len(elements) > 0 {
        text, _ := elements[0].Text()
        if strings.Contains(text, "Success") {
            return nil
        }
    }
}
```

Step Implementation (continued)

```go
func stepSeeShortenedURL() error {
    links, _ := testCtx.webDriver.FindElements(selenium.ByID, "shortened")
    for _, link := range links {
        href, _ := link.GetAttribute("href")
        if strings.Contains(href, "http://localhost:8080/") && !strings.HasSuffix(href, "/") {
            parts := strings.Split(href, "/")
            testCtx.lastShortCode = strings.TrimSpace(parts[len(parts)-1])
            return nil
        }
    }

    return fmt.Errorf("shortened URL not found")
}
```

Test Execution Flow

What Happens When You Run `go test ./...`

1. **Build**: Application binary is compiled
2. **Start GeckoDriver**: GeckoDriver runs on port 4444
3. **Start Server**: Application runs on localhost:8080
4. **Parse Features**: Godog reads `.feature` files
5. **For each scenario**:
   - Execute `Given` steps (setup)
   - Execute `When` steps (actions)
   - Execute `Then` steps (assertions)
6. **Cleanup**: Stop GeckoDriver, server. remove test database.
7. **Report**: Display pass/fail results

Test Output Example

Console Output

```
...
...
...

21 scenarios (21 passed)
135 steps (135 passed)
```

Benefits of This Approach

Why BDD with Godog?

1. Readable Tests

- Non-technical stakeholders can understand
- Feature files serve as documentation

2. Behavior-Focused

- Tests describe user behavior, not implementation
- Tests remain stable even if code changes

3. Comprehensive Coverage

- Hash function testing
- Normalization verification
- Collision handling
- Input validation

Integration Benefits

4. Real Browser Testing

- Uses actual Firefox browser
- Tests real user interactions
- Catches UI/UX issues

5. Automated Everything

- Server starts automatically
- GeckoDriver managed by test code
- Clean setup and teardown

6. CI/CD Ready

- Runs in GitHub Actions
- Generates coverage reports
- Publishable test results

Test Statistics

Test Coverage

7 Scenario Types, 21 Test Cases

Category	Scenarios	Purpose
Valid URLs	2	Hash generation
Normalization	4	URL normalization
Duplicates	2	Collision handling
Special chars	2	Encoding
Invalid URLs	7	Validation
Non-existent	2	Error handling
Format validation	2	Compliance

Total execution time: ~10-30 seconds

Automate Test Execution

Github Actions

```yaml
...

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install Rust and Cargo
        uses: actions-rs/toolchain@v1
        with:
          toolchain: stable
          override: true
      - name: Install Geckodriver via Cargo
        run: |
          cargo install geckodriver
      - name: Set up Go
        uses: actions/setup-go@v4
        with:
          go-version: "1.25.4"
      - name: Test
        run: |
          cd test
          go test -v
```

Questions?

Project Repository

GitHub: https://github.com/ItsDobiel/URLShortener

Documentation:

- `README.md` - Project overview
- `presentation/contents.md` - This presentation

Try it yourself:

```
git clone https://github.com/ItsDobiel/URLShortener.git
cd urlshortener
make test ./...
```

The End!

Resources

Course: Software Testing

Technologies Used for Testing:

- Go
- Godog (BDD framework)
- Selenium
- GeckoDriver

References:

- Godog: https://github.com/cucumber/godog
- Gherkin: https://cucumber.io/docs/gherkin