Smart URL Shortener with Custom Aliases and Expiry Management

PROJECT REPORT

Submitted by

R. JUPITER	**REGISTER. NO: 23TD0341**
P. SIVA RAJ	**REGISTER. NO: 23TD0402**
S. SUKHESH RAM	**REGISTER. NO: 23TD0406**

Under the Guidance of

Ms. D. MOHANA PRIYA, M.Tech,

Assistant Professor

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MANAKULA VINAYAGAR INSTITUTE OF TECHNOLOGY

KALITHEERTHALKUPPAM, PUDUCHERRY- 605 107

PONDICHERRY UNIVERSITY

INDIA

MAY - 2025

Table of Contents

ABSTRACT

In the digital era, long and complex URLs can hinder the ease of sharing and communication. This project introduces a Smart URL Shortener, a full-stack web application designed to convert lengthy URLs into short, manageable links with advanced features such as custom aliases, expiry control, and click tracking. The backend is developed using Java (Spring Boot) and MongoDB, while the frontend utilizes React.js for a modern, responsive user interface. A Time-To-Live (TTL) index is used in the database to automatically handle link expiration. While user interaction and real-time countdown are provided on the frontend, analytics like click count tracking are handled in the backend and stored in the database. These analytics are currently not visualized on the frontend UI. This modular design allows for future integration of complete analytics dashboards and QR code generation features. Overall, the project offers a secure, customizable, and extensible solution to simplify URL sharing.

1. Introduction

1.1. Background

In With the exponential growth of the internet and the need to share web content quickly, long and complex URLs have become a challenge. Whether it’s social media, messaging apps, or printed material, users often need clean, concise links. URL shorteners solve this issue by converting lengthy URLs into compact, manageable versions that are easier to distribute and remember.

While many commercial platforms offer URL shortening services, most come with limitations such as lack of customization, restricted analytics, or payment walls. Motivated by the need for a flexible, open-source alternative, this project was designed to provide a customizable and intelligent solution that adds extra features like custom aliases, link expiration, and the potential for analytics — all built from scratch using a modern tech stack.

1.2. Problem Statement

Users often struggle with sharing long, unattractive URLs, especially on platforms with character limitations or aesthetic constraints. Existing tools either lack customization or require paid plans to access advanced features. There is a need for a lightweight, user-friendly URL shortening service that:

- Allows users to shorten URLs efficiently
- Enables custom alias assignment
- Supports time-based expiration
- Tracks link usage (click count)

This project addresses these issues by building a fully functional URL shortener using modern web development technologies.

1.3. Objective

This mini project aims to develop a fully functional URL shortener that simplifies sharing long web links. It provides users with a clean interface to generate short URLs and supports custom aliases for personalized or branded links. Key features include automatic link expiration using MongoDB's TTL indexing and click tracking to monitor link usage. The frontend is built to be user-friendly and responsive, offering smooth interactions and real-time link expiry updates for an enhanced user experience.

2. Proposed Work

2.1. System Overview

The proposed URL shortener system is a full-stack web application designed to generate shortened versions of long URLs with additional features such as custom aliases, link expiration, and click tracking. It is built using a modern technology stack: Spring Boot and Java for the backend, MongoDB for persistent storage, and React.js for the frontend interface. The backend provides RESTful APIs that handle URL creation, redirection, and analytics, while the frontend offers users a clean, interactive UI for input and feedback. The system is designed to be scalable, modular, and easy to extend in future versions.

2.2. Functional Requirements

- Users Users must be able to submit long URLs to receive shortened versions.
- Users can optionally specify a custom alias.
- Each shortened link should redirect users to the original URL.
- The system must track the number of times each short URL is clicked.
- Users can specify an expiry date and time for each shortened link.
- Expired URLs should no longer redirect and should be automatically removed by the database.
- Remaining time until expiration should be displayed on the frontend.
- A toast message should confirm when a shortened link is copied.

2.3.Non-Functional Requirements

- The system should be responsive and provide a smooth user experience.
- APIs should respond with proper HTTP status codes and messages.
- Data should be stored efficiently and securely in MongoDB.
- The application must allow CORS access only from trusted origins (e.g., the frontend).
- The system should be capable of handling concurrent requests gracefully.

2.4.Modules Description

- URL Submission Module: Takes user input (long URL, custom alias, expiry) and sends it to the backend.
- URL Shortening Logic: Handles creation of short URLs using either a random ID or user-defined alias.
- Redirection Handler: Resolves the short URL and redirects users to the original link.

- Expiration Checker: Compares the current time with the stored expiry time and disables expired links.
- Click Tracker: Increments the click count for every access.
- Frontend Interface: Displays the form, result, error messages, and countdown timer.
- Clipboard & Toast Module: Lets users copy the short URL and confirms the action via a toast popup.

2.5.Data Flow / Workflow

1. The user enters a long URL, custom alias (optional), and expiry time (optional) into the React form.
2. The frontend sends a POST request to the Spring Boot backend via /api/shorten.
3. The backend checks the alias (if given), generates a short URL, stores it in MongoDB with metadata.
4. MongoDB automatically deletes the link when the expiresAt field is reached (using TTL index).
5. When a short URL is accessed, the /api/{shortUrl} endpoint redirects to the original URL after verifying it is not expired.
6. The system increments the click count and can return expiry status using another API call.
7. Frontend fetches and displays the shortened URL, expiry time, and copy option.

2.6.Use Case Scenarios

- Use Case 1 – Basic URL Shortening: A user enters a long link, and the system returns a shortened version.
- Use Case 2 – Custom Alias: A user inputs a preferred alias; the backend checks for conflicts and stores it.
- Use Case 3 – Expiry Management: The user sets an expiry date/time, and after expiration, the link becomes inactive.
- Use Case 4 – Link Access and Tracking: A visitor clicks on a short link, is redirected, and the click count is recorded.
- Use Case 5 – Frontend Interaction: The user copies the short URL and sees a toast confirmation, along with a real-time countdown.

3. Architecture

3.1. System Architecture Overview

The system is designed using a layered architecture to ensure separation of concerns, modularity, and maintainability. It follows a client-server model, where:

- The frontend (client) is built using React.js and handles all user interactions.
- The backend (server) is built with Spring Boot and provides REST APIs.
- The MongoDB database serves as the persistent storage for all URL data, including original URLs, shortened aliases, expiration times, and click counts.

3.2. Component-Based Architecture

Here are the key components involved:

- Frontend (React.js)
  - Collects user input (URL, alias, expiration).
  - Sends API requests to the backend.
  - Displays shortened URL, remaining time, and copy functionality.
- Backend (Spring Boot)
  - Exposes RESTful endpoints to create and retrieve URLs.
  - Performs business logic like alias checking, link generation, and expiration validation.
  - Tracks usage (click counts) and manages link lifecycle.
- Database (MongoDB)
  - Stores all URL documents.
  - Uses TTL indexing to delete expired URLs automatically.

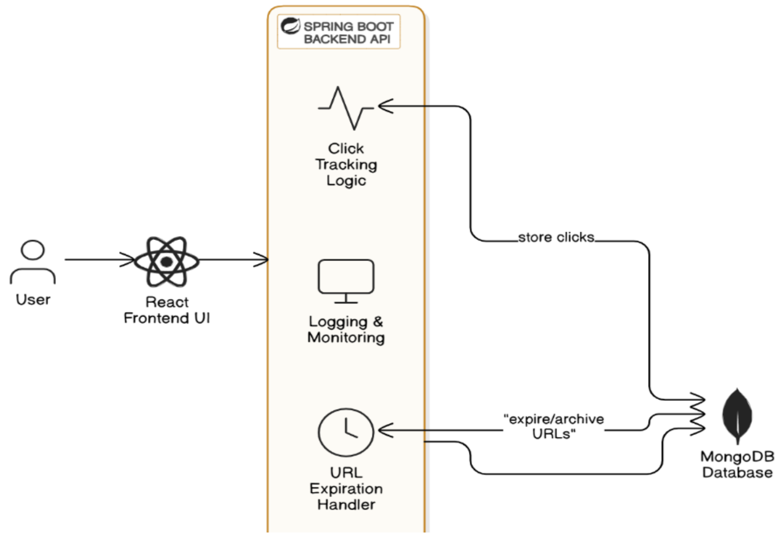


Figure 1:System Architecture of the URL Shortener

3.3. Data Flow Sequence

- User submits URL + alias + expiry.
- React sends a POST request to /api/shorten.
- Spring Boot processes and stores data in MongoDB.
- On redirection, Spring Boot checks expiration and redirects using stored URL.
- Click count is updated in MongoDB.
- React can query /remaining-time/{shortUrl} to get and display expiry countdown.

4. Code Used

4.1. Backend – UrlController.java

```
@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "http://localhost:3000")
public class UrlController {

    private final UrlService urlService;

    public UrlController(UrlService urlService) {
        this.urlService = urlService;
    }

    @PostMapping("/shorten")
    public ResponseEntity<?> shortenUrl(@RequestBody UrlRequest request) {
        Url shortenedUrl = urlService.saveUrl(
            request.getOriginalUrl(),
            request.getCustomAlias(),
            request.getCustomDomain(),
            request.getExpiresAt()
        );
        return ResponseEntity.ok(shortenedUrl);
    }

    @GetMapping("/{shortUrl}")
    public ResponseEntity<?> redirect(@PathVariable String shortUrl) {
        Optional<Url> urlOptional = urlService.getUrl(shortUrl);

        if (urlOptional.isPresent()) {
            Url url = urlOptional.get();
            if (url.getExpiresAt() != null && url.getExpiresAt().isBefore(Instant.now())) {
                return ResponseEntity.status(410).body("This link has expired!");
            }
            urlService.incrementClickCount(url);
            return ResponseEntity.status(302).location(URI.create(url.getOriginalUrl())).build();
        }

        return ResponseEntity.notFound().build();
    }

    @GetMapping("/remaining-time/{shortUrl}")
    public ResponseEntity<?> getRemainingTime(@PathVariable String shortUrl) {
        long remainingTime = urlService.getTimeRemainingBeforeExpiration(shortUrl);
        if (remainingTime > 0) {
            return ResponseEntity.ok("Time remaining before expiration: " + remainingTime + "
milliseconds");
        } else {
            return ResponseEntity.status(410).body("The link has expired or does not exist.");
        }
    }
}
```

4.2.Backend – Url.java

```
package com.Java_mini_project.url_shortener.model;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

import java.time.Instant;

@Data
@Document(collection = "urls")
public class Url {
    @Id
    private String shortUrl;
    private String originalUrl;
    private Instant createdAt;

    @Indexed(expireAfterSeconds = 0)
    private Instant expiresAt;
    private long clickCount;
    private String customDomain;
}
```

4.3.Backend – UrlRequest.java

```
package com.Java_mini_project.url_shortener.model;

import java.time.Instant;

public class UrlRequest {
    private String originalUrl;
    private String customAlias;
    private String customDomain;
    private Instant expiresAt;

    public String getOriginalUrl() {
        return originalUrl;
    }

    public void setOriginalUrl(String originalUrl) {
        this.originalUrl = originalUrl;
    }

    public String getCustomAlias() {
        return customAlias;
    }

    public void setCustomAlias(String customAlias) {
        this.customAlias = customAlias;
    }

    public String getCustomDomain() {
        return customDomain;
    }

    public void setCustomDomain(String customDomain) {
        this.customDomain = customDomain;
    }

    public Instant getExpiresAt() {
        return expiresAt;
    }

    public void setExpiresAt(Instant expiresAt) {
        this.expiresAt = expiresAt;
    }
}
```

4.4.Backend – UrlRepository.java

```
package com.Java_mini_project.url_shortener.repository;


import com.Java_mini_project.url_shortener.model.Url;
import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.Optional;


public interface UrlRepository extends MongoRepository<Url, String> {
    Optional<Url> findByShortUrl(String shortUrl);
}
```

4.5.Backend – DeletionLog.java

```
package com.Java_mini_project.url_shortener.model;


import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.time.Instant;


@Data
@Document(collection = "deletion_logs")
public class DeletionLog {
    @Id
    private String id;
    private String shortUrl;
    private String originalUrl;
    private Instant expiredAt;
    private Instant deletedAt;
}
```

4.6.Backend – DeletionLogRepository.java

```
package com.Java_mini_project.url_shortener.repository;

import com.Java_mini_project.url_shortener.model.DeletionLog;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface DeletionLogRepository extends MongoRepository<DeletionLog, String> {
}
```

4.7.Backend – UrlService.java

```
package com.Java_mini_project.url_shortener.service;

import com.Java_mini_project.url_shortener.model.Url;
import com.Java_mini_project.url_shortener.model.DeletionLog;
import com.Java_mini_project.url_shortener.repository.UrlRepository;
import com.Java_mini_project.url_shortener.repository.DeletionLogRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.Instant;
import java.util.List;
import java.util.Optional;
import java.util.Random;
```

```
@Service
public class UrlService {
    private static final Logger logger = LoggerFactory.getLogger(UrlService.class);
    private final UrlRepository urlRepository;
    private final DeletionLogRepository deletionLogRepository;

    private static final String CHARACTERS =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    private static final int SHORT_URL_LENGTH = 6;

    public UrlService(UrlRepository urlRepository, DeletionLogRepository
deletionLogRepository) {
        this.urlRepository = urlRepository;
        this.deletionLogRepository = deletionLogRepository;
    }

    public String generateShortUrl() {
        Random random = new Random();
        StringBuilder sb = new StringBuilder(SHORT_URL_LENGTH);
        for (int i = 0; i < SHORT_URL_LENGTH; i++) {
            sb.append(CHARACTERS.charAt(random.nextInt(CHARACTERS.length())));
        }
        return sb.toString();
    }

    public Url saveUrl(String originalUrl, String customAlias, String customDomain, Instant
expiresAt) {
        String shortUrl = (customAlias != null && !customAlias.isEmpty()) ? customAlias :
generateShortUrl();
```

```
    if (customDomain == null || customDomain.isEmpty()) {
        customDomain = "http://localhost:8080";
    }

    if (urlRepository.findByShortUrl(shortUrl).isPresent()) {
        throw new RuntimeException("Custom alias already taken!");
    }

    if (expiresAt == null) {
        expiresAt = Instant.now().plusSeconds(7 * 24 * 60 * 60);
    }

    Url url = new Url();
    url.setShortUrl(shortUrl);
    url.setOriginalUrl(originalUrl);
    url.setCreatedAt(Instant.now());
    url.setExpiresAt(expiresAt);
    url.setClickCount(0);
    url.setCustomDomain(customDomain);

    return urlRepository.save(url);
}

public Optional<Url> getUrl(String shortUrl) {
    Optional<Url> urlOptional = urlRepository.findByShortUrl(shortUrl);

    if (urlOptional.isPresent()) {
        Url url = urlOptional.get();
        if (url.getExpiresAt() != null && url.getExpiresAt().isBefore(Instant.now())) {
            logger.warn("Short URL '{}' expired at {}", shortUrl, url.getExpiresAt());

            DeletionLog log = new DeletionLog();
            log.setShortUrl(url.getShortUrl());
            log.setOriginalUrl(url.getOriginalUrl());
            log.setExpiredAt(url.getExpiresAt());
            log.setDeletedAt(Instant.now());
            deletionLogRepository.save(log);

            urlRepository.delete(url);

            return Optional.empty();
        }
        return Optional.of(url);
    }

    return Optional.empty();
}

public void incrementClickCount(Url url) {
    url.setClickCount(url.getClickCount() + 1);
    urlRepository.save(url);
}

@Scheduled(fixedRate = 3600000)
@Transactional
```

```
    public void deleteExpiredLinks() {
        List<Url> expiredUrls = urlRepository.findAll().stream()
            .filter(url -> url.getExpiresAt() != null && url.getExpiresAt().isBefore(Instant.now()))
            .toList();

        for (Url url : expiredUrls) {
            logger.warn("Deleting expired short URL: {}", url.getShortUrl());
            urlRepository.delete(url);
        }
    }

    public long getTimeRemainingBeforeExpiration(String shortUrl) {
        Optional<Url> urlOptional = urlRepository.findByShortUrl(shortUrl);
        if (urlOptional.isPresent()) {
            Url url = urlOptional.get();
            Instant expiresAt = url.getExpiresAt();

            if (expiresAt != null) {
                long timeRemaining = expiresAt.toEpochMilli() - Instant.now().toEpochMilli();
                return timeRemaining > 0 ? timeRemaining : 0;
            }
        }
        return 0;
    }
}
```

4.8.Frontend – UrlShortener.js

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const requestData = {
      originalUrl,
      customAlias,
      expiresAt: expiresAt ? new Date(expiresAt).toISOString() : null,
    };

    try {
      const response = await axios.post("http://localhost:8080/api/shorten", requestData);
      const fullShortUrl = `http://localhost:8080/api/${response.data.shortUrl}`;
      setShortenedUrl(fullShortUrl);
      fetchRemainingTime(response.data.shortUrl);
      setError("");
    } catch (err) {
      setError("Error shortening the URL");
      console.error(err);
    }
  };

  const fetchRemainingTime = async (shortUrl) => {
    try {
      const response = await axios.get(`http://localhost:8080/api/remaining-time/${shortUrl}`);
      setTimeRemaining(parseInt(response.data.match(/\d+/)[0]));
    } catch (err) {
      setTimeRemaining(null);
      console.error("Error fetching time remaining:", err);
    }
  };

  const formatTime = (ms) => {
    if (ms <= 0) return "Expired";
    const days = Math.floor(ms / (1000 * 60 * 60 * 24));
    const hours = Math.floor((ms % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
    const minutes = Math.floor((ms % (1000 * 60 * 60)) / (1000 * 60));
    const seconds = Math.floor((ms % (1000 * 60)) / 1000);
    return `${days}d ${hours}h ${minutes}m ${seconds}s`;
  };.
```

4.9. Explanation of Key Logic

- URL Shortening: UrlController receives original URL and optional custom alias/expiry. It stores data in MongoDB and returns a short link.
- Redirection: If the link is valid and not expired, the user is redirected to the original URL.

- Link Expiry: Uses MongoDB TTL to delete expired links automatically.
- Time Remaining: A helper API checks how long a link is still valid and returns remaining milliseconds.
- Frontend Interaction: React form handles input, calls the backend API, and shows the short link along with expiry countdown.

5. Screenshot Explanation and UI Functionality Overview

5.1. URL Shortening Interface

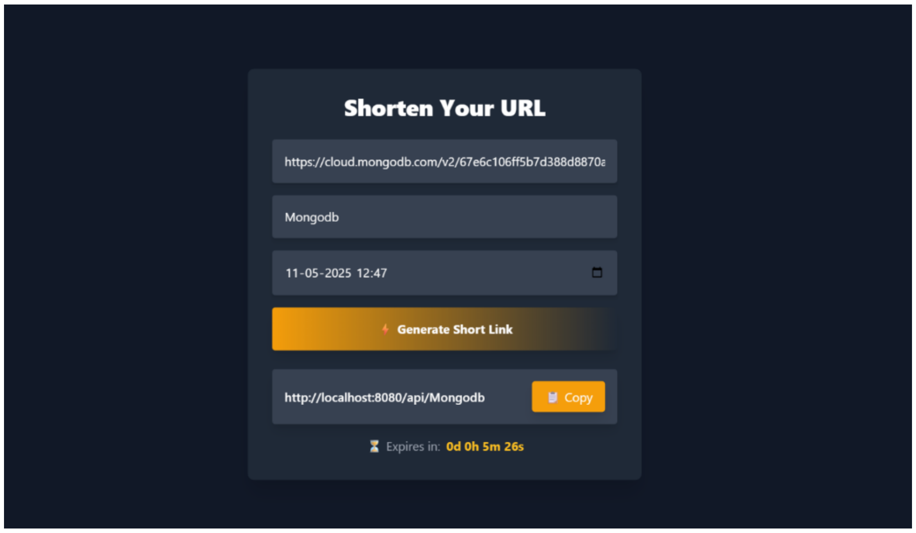


Figure 2: URL Shortening Interface

UI Elements Present:

- Original URL input: Allows users to input long URLs (e.g., https://cloud.mongodb.com/...)
- Custom alias input: Enables users to create custom short aliases (e.g., Mongodb)
- Expiration datetime picker: A tool to select the expiration date and time for the shortened URL (e.g., 11-05-2025 12:47)
- Generate Short Link button: A stylish button with gradient and animation to generate the shortened URL
- Output section: Displays the shortened URL (e.g., http://localhost:8080/api/Mongodb)
- Copy button: Allows users to copy the shortened URL to their clipboard
- Countdown timer: Dynamically shows the time remaining until the link expires.

Highlights:

- Modern UI: The interface is designed using a dark theme, TailwindCSS classes, and smooth animations.
- Functional Countdown: The countdown timer works effectively and updates every second, providing real-time expiration information.

5.2. Time Remaining Checker

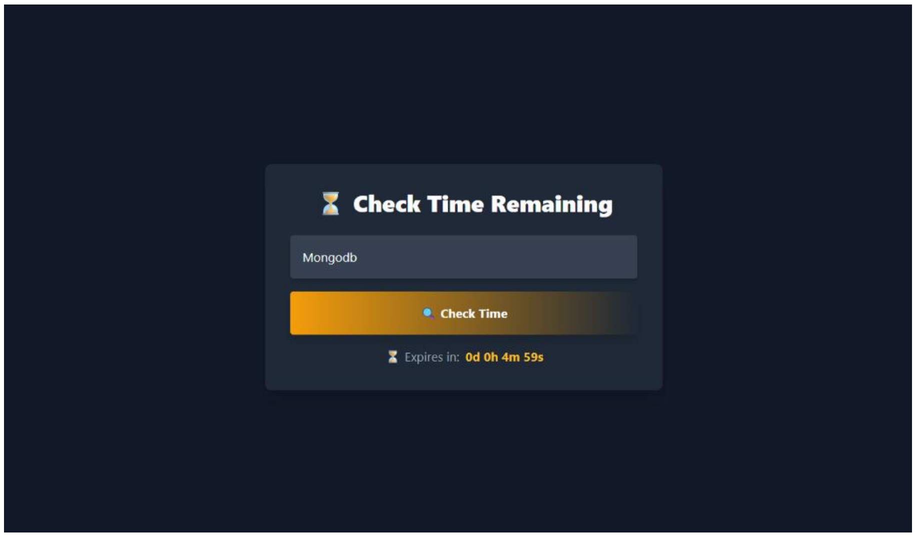


Figure 3: Time Remaining Checker

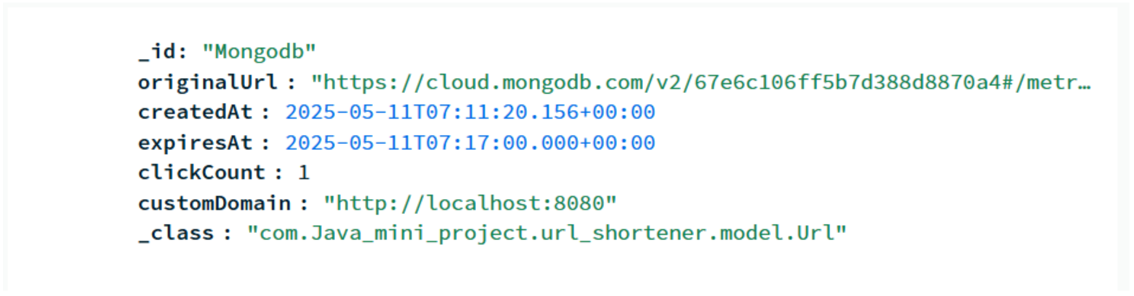
UI Elements Present:

- Custom alias input: Users can enter the alias of a shortened URL.
- Check Time button: A button to trigger the check for time remaining before expiration.
- Remaining time display: Shows the exact remaining time in a format such as "Expires in: 0d 0h 4m 59s".

Highlights:

- Clean UI: The interface is minimalistic, offering a smooth user experience with visual feedback.
- Real-Time Information: Displays the remaining time dynamically by fetching data from the backend API (/remaining-time/{shortUrl}).

5.3. MongoDB Document



```
_id: "Mongodb"
originalUrl : "https://cloud.mongodb.com/v2/67e6c106ff5b7d388d8870a4#/metr…
createdAt : 2025-05-11T07:11:20.156+00:00
expiresAt : 2025-05-11T07:17:00.000+00:00
clickCount : 1
customDomain : "http://localhost:8080"
_class : "com.Java_mini_project.url_shortener.model.Url"
```

Figure 4:MongoDB Document

- Stored Data for Alias Mongodb:
- originalUrl: Stores the full original URL, such as https://cloud.mongodb.com/....
- createdAt: Timestamp when the shortened URL was created (e.g., 2025-05-11T07:11:20.156+00:00).
- expiresAt: Expiration date and time (e.g., 2025-05-11T07:17:00.000+00:00), which corresponds to the countdown timer.
- clickCount: Tracks how many times the shortened URL has been accessed (e.g., clickCount: 1).
- customDomain: Stores the custom domain used for the shortened URL (e.g., http://localhost:8080).
- Highlights:
- TTL Index: MongoDB's TTL (Time-To-Live) index ensures URLs are automatically deleted after expiration.
- Accurate Data Mapping: The backend correctly maps data to the Url.java model, ensuring consistency in the database.

6. Challenges & Learnings

6.1. Challenges

- Integration Issues: Faced challenges with integrating the frontend and backend, particularly around ensuring smooth communication between the React UI and the Spring Boot API.
- Time Handling: Implementing and managing the URL expiration time proved complex, especially handling time zones and ensuring accurate countdown functionality.
- CORS: Dealt with Cross-Origin Resource Sharing (CORS) issues, which required proper configuration to allow communication between the frontend and backend on different ports.

6.2. Learnings

- Gained experience in full-stack development, understanding both frontend (React.js) and backend (Spring Boot) components.
- Implemented MVC (Model-View-Controller) architecture to structure the code effectively.
- Learned how to leverage MongoDB's TTL (Time-to-Live) indexing to automatically expire URLs.
- Focused on improving frontend UX with a smooth, intuitive interface.

7. Conclusion

7.1.Recap of Achievement

Successfully developed a URL shortener application that efficiently shortens URLs, supports custom aliases, handles URL expiration, and tracks clicks. The application is equipped with a modern, responsive UI and backend integration using Spring Boot and MongoDB.

7.2.Future Enhancements

- QR Code Generation: Adding QR code functionality for easy sharing.
- Better Analytics: Introducing detailed click analytics, including user location, device type, etc.
- Link Preview: Providing a preview of the original URL content before redirection.

8. References

- Spring Boot Documentation – https://spring.io/projects/spring-boot
- React.js Documentation – https://reactjs.org/docs/getting-started.html
- MongoDB TTL Indexing – https://www.mongodb.com/docs/manual/core/index-ttl/
- Axios Library – https://axios-http.com/
- TailwindCSS – https://tailwindcss.com/docs
- CORS in Spring Boot – https://spring.io/guides/gs/rest-service-cors/
- Java Date and Time API (java.time) – https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html