# CAPSTONE PROJECT SRS

# (SOFTWARE REQUIREMENT SPECIFICATIONS)

# PROJECT TITLE – URL SHORTNER

# GROUP – 10

**Team Members**

11626 - Saathwik.G

11627 - Dinesh.V

11613 - Sabiha.Sk

11628 - Afreen Taj

# Chapter 1: Introduction

## 1.1 Purpose

The purpose of this document is to outline the requirements and specifications for the development of a URL shortener. The URL shortener aims to provide users with a convenient and efficient way to generate short URLs from long ones, facilitating easy sharing and tracking of links. This document serves as a comprehensive guide for the development team, outlining the functional and non-functional aspects of the URL shortener to ensure a successful and well-defined implementation.

## 1.2 Scope

The system will provide users with the ability to generate shortened URLs from long ones. Key functionalities include URL shortening, redirection, and basic analytics. The project aims to deliver a user-friendly and efficient solution for link management. However, it does not extend to the development of advanced features such as custom short URLs, deep analytics, or user accounts. The focus is on creating a streamlined and reliable URL shortening service.

## 1.3 Definition, Acronyms & Abbrevations

- **URL:** URL stands for Uniform Resource Locator.
- **Hash Functions:** A hash function is a quick and deterministic algorithm that converts input data into a fixed-size string of characters, ensuring unique outputs for different inputs. It is widely used for data integrity, password storage, and cryptographic purposes.
- **DNS:** DNS stands for Domain Name System.

## 1.4 References


- **Java Documentation:** Java Documentation - Get Started (oracle.com)
- **Spring Boot Documentation:** Spring Boot Reference Documentation
- **MySQL Documentation:** https://dev.mysql.com/doc/
- **React JS Documentation:** https://reactjs.org/docs/getting-started.html

## 1.5 Overview

"This project is designed to streamline the process of sharing and managing links. The project focuses on providing users with a straightforward solution to convert lengthy URLs into concise and user-friendly short links. This web-based application facilitates easy sharing on platforms with character limitations and enhances the overall user experience.

Key functionalities include the ability to submit a long URL and receive a shortened version, as well as a seamless redirection mechanism for users clicking on the generated short links. The project emphasizes simplicity, efficiency, and reliability in its core features, aiming to meet the fundamental need for efficient URL management without unneces

# Chapter 2: Overall Description

## 2.1 Product Perspective

From a user's perspective, this serves as a reliable and accessible service for transforming and managing URLs. It simplifies the process of sharing links by providing concise alternatives. The system's independence allows it to be easily incorporated into different environments, enhancing the overall user experience without imposing extensive integration requirements.

## 2.2 Product Functions

- **URL Shortening:** Users can submit long URLs to the system.
- The system generates a shortened URL for each submitted long URL.
- **URL Redirecting:** Users clicking on a shortened URL are seamlessly redirected to the original, longer URL.
- **Analytics (Basic):** Basic tracking of link usage, including the number of clicks on each shortened URL.
- **User Interaction:** User interaction refers to the exchange of information and actions between a person and a system, often involving input, feedback, or communication.
- **User-friendly Interface:** A straightforward and intuitive web-based user interface for submitting and managing URLs.

## 2.3 Users Classes and Characteristics

**Regular users:** Regular users have authentication to login and access the services.

## 2.4 Design and implementation constraints

- Each longURL must be hashed to a unique hashValue even when the servers scale horizontally. To address this, a distributed unique id generator can be used to generate unique ids across the cluster.
- A potential security problem we could face is that malicious users send an overwhelmingly large number of URL shortening requests. Rate limiter helps to filter out requests based on IP address or other filtering rules.

## 2.5 Assumptions and Dependencies

**Assumptions:**

- A tinyurl will be stored in the server for 180 days.
- The hash algorithm generates unique code for each long url.
- The hash code is of length 6 characters.

### 2.5.1.Dependencies:

**Technology Dependencies:** The technology dependencies for a URL shortener project include various components spanning both the backend and frontend, as well as infrastructure and deployment.

**Data Source Integration:** These dependencies are crucial for handling the complexities of integrating external data sources into a URL shortener. Consider the specific requirements of your project, such as the types of data sources you plan to integrate and the security measures needed for handling sensitive information.

**Change Management:** Effective change management is critical for maintaining the stability, reliability, and security of a URL shortener project as it evolves. These dependencies help streamline the process, mitigate risks, and ensure successful changes to the system.

# Chapter 3: Specific Requirements

## 3.1 Functional Requirements

API Endpoints: API endpoints facilitate the communication between clients and servers. We will design the APIs REST-style. A URL shortener primary needs two API endpoints.
**1. URL shortening:** To create a new short URL, a client sends a POST request, which contains one parameter: the original long URL.
The API looks like this: POST api/v1/data/shorten
request parameter: {longUrl: longURLString} return shortURL.

**2. URL redirecting:** To redirect a short URL to the corresponding long URL, a client sends a GET request.
The API looks like this: GET api/v1/shortUrl
Return longURL for HTTP redirection

3. **HTTP Redirection:** Once the server receives a tinyurl request, it changes the short URL to the long URL with 301 redirect.

## 3.2 Database Requirements

**3.2.1 User Database:** Data base Stores users login credentials

**3.2.2 Database Structure:** The database structure for URL shortening typically involves creating tables to store information related to original long URLs, shortened codes, and any additional metadata needed for tracking and managing links.

## 3.3 Performance Requirements

**Low Latency:** Swift redirection for minimal wait times.

**High Availability:** Uninterrupted service accessibilility for users.

**High Scability:** Efficient handling of growing demand without performance degradation.

## 3.4 Software Quality Attributes

**3.4.1 Reliabilty:** The system's ability to perform consistently and accurately under varying conditions, ensuring data integrity, user authentication, and reliable functionality throughout user interactions.

**3.4.2 Data Integrity:** Implement measures to ensure the integrity of data stored in the

database, preventing data corruption and ensuring the reliability of URL mappings.

## 3.5 Software & Hardware Requirements

### 3.5.1 Software Requirements

**Frontend Development: HTML, CSS** and **React.js** is utilized in URL shortener frontend development for efficient, modular, and dynamic user interfaces, enabling seamless interactions and improved user experience.

**Backend Development: Spring Boot** enables developers to build robust applications having secure as well as clear configurations without losing much time and effort on itscomplex framework.

**Database ManagementSystem: MySQL i**s used to store and manage the URL data.

**Version Control: Git** is used for version control, enabling collaborative development, tracking changes, and managing code base versions.

**DNS Service :** DNS service is used in URL shortner to provide human readable custom domain addresses ,enhancing brand identity and user recognition for shortened URLs

**Domain name :** Domain names are used in URL shorteners to offer recognizable, branded, and user friendly addresses, enhancing the accessibility and trustworthiness of shortened URLs.

**Continuous Integration/Continuous Deployment (CI/CD):** Jenkins is used in integrated into the development workflow to automate the CI/CD pipeline for efficient development,testing, and deployment processes.

**Infrastructure and cloud services**: route 53, elastic kubernetes services and application load balancer.

### 3.5.2 Hardware Requirements

**Development Machines:** Standard development machines with sufficient RAM (8GB ormore), modern processors, and ample storage.

**Build and Deployment Servers:** Servers with sufficient resources for building anddeploying the application.

**Database Server:** Dedicated server or cloud-based instance with adequate resources forhosting the MySQL database.