**Project Report on**



**URL SHORTENER USING DJANGO**



**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Description** | **Page No.** |
| Chapter 1 | Introduction | 1 |
| Chapter 2 | Literature Survey | 3 |
| Chapter 3 | Methodology | 5 |
| Chapter 4 | Result and Discussion | 9 |
| Chapter 5 | Conclusion and Future Work | 12 |
|  | References | 14 |

**Chapter 1**

**Introduction**

* 1. **Introduction**

URL Shortener: Making Long Web Addresses Shorter

Hello and welcome to the realm of URL shortening! Long and lengthy URLs can be a burden in today's fast-paced digital world, where sharing links has become a vital part of our lives. This is when a URL shortener comes in handy.

In the enormous world of the internet, we frequently encounter large, bulky URLs that are difficult to share, remember, or use efficiently. URL shorteners come in handy here, providing a quick way to simplify and streamline long web addresses.

A URL shortener is an application that converts a long URL into a condensed, compact version that is easier to share and manage. When the shortened URL is clicked or viewed, it redirects people to the original, lengthier URL. Because of its capacity to improve the user experience and maximize digital communication, this technology has become extremely popular.

This is when URL shorteners come in handy. A URL shortener is a simple but effective tool for converting long URLs into shorter, more manageable links. The idea behind these services is simple: they take the original, often cumbersome URL and convert it into a shorter, more user-friendly version. A long URL, such as "https://www.example.com/articles/how-to-use-url-shortener-for-beginners", can be abbreviated to https://short.link/xyz123.

A URL shortener's primary purpose is to make URLs more manageable, aesthetically beautiful, and adaptable to various internet platforms. A shortened URL eliminates clutter, improves readability, and increases clickability when sharing links on social media, sending URLs via messaging applications, or embedding them in emails.

URL shorteners provide several benefits in addition to shortening URLs. They frequently include useful capabilities such as tracking click data, monitoring engagement metrics, and analysing user behaviour.

This information can assist marketers, firms, and individuals in determining the efficiency of their online initiatives, measuring audience reach, and refining their strategy as needed.

Furthermore, by allowing users to generate bespoke, customized links, URL shorteners can improve branding and recognition. Individuals and organizations can create a unified online presence by including their names, keywords, or distinctive expressions into shortened links instead of utilizing generic, impersonal URLs.

URL shorteners are useful tools for simplifying, optimizing, and improving the use of URLs in a variety of online scenarios. URL shorteners enable customers to navigate the digital world more effectively and make the most of their online experiences by compressing lengthy site addresses, offering data insights, and enabling personalised branding.

To summarize, a URL shortener is a useful tool for converting long web addresses into short and memorable links. It simplifies the sharing process, improves the user experience, and provides useful analytics. A URL shortener is your go-to answer if you're a social media enthusiast, advertiser, or simply someone trying to conserve characters. So, prepare to simplify your URLs and discover the world of URL shortening!

**Chapter 2**

**Literature Survey**

This survey of the literature attempts to evaluate the many research on URL shorteners that has been conducted in a variety of domains, including computer science, information security, web technology, social media, and marketing. We want to understand the methods, benefits, challenges, and implications of using URL shorteners, thus we'll synthesize the existing material on the subject.

**2.1 URL Shortener Algorithms & Techniques:**

The algorithms and strategies used to generate short URLs are the topic of this research. To generate compact and unique short URLs, studies have proposed hash-based, bijective mapping, and other algorithms. To ensure reliable redirections, efforts have been taken to strengthen the collision resistance of produced URLs.

**2.2 Performance and Scalability:**

Researchers explored the performance and scalability of URL shorteners in response to the increasing demand for these services. Studies are being conducted to investigate strategies for efficiently handling huge amounts of short URLs while ensuring low-latency redirections even during peak usage.

**2.3 Concerns about security and privacy:**

A substantial corpus of research has been conducted into the security and privacy implications of URL shorteners. Potential concerns noted by researchers include phishing attacks, virus dissemination, and the surveillance of users' internet behavior for targeted advertising. To combat these dangers, proposed solutions include verification processes, link previews, and other security measures.

**2.4 Adoption and User Behaviour:**

Understanding user behaviour and the use of URL shorteners is an important topic of research. The factors that influence users' decisions to click on shorter links, user perceptions of security and trust, and the impact of shortened URLs on user engagement have all been studied.

**2.5 Link Decay and Longevity:**

The phenomenon of link decay has gained attention, in which shortened URLs become invalid over time due to link rot. Studies on the lifespan of short URLs have been conducted, and measures to prevent link decay, such as monitoring and changing destination URLs, have been recommended.

**2.6 Social media and Marketing Applications:**

The usage of URL shorteners in social media and marketing initiatives has received a lot of attention. The study looks at the usefulness of abbreviated URLs in generating user engagement, click-through rates, and marketing statistics.

**2.7 Legal and Ethical Considerations:**

Some research have looked into the legal and ethical implications of URL shorteners. These factors include privacy concerns, data collecting and utilization, and URL shortener service providers' obligations in preventing abuse.

**2.8** **URL Shortener Services and Trends:**

Comparative studies have been conducted to examine the features, performance, and user satisfaction of various URL shortener services. Furthermore, the research focuses on emerging trends in URL shorteners, such as the incorporation of analytics, QR codes, and customized short domains.

**Chapter 3**

**Methodology**

We utilized Django for this project, and HTM (Hypertext Markup Language) and CSS (Cascading Style Sheet) for the frontend. And for databases, we have SQLite (for data management).

Django is a Python-based free and open-source web framework that follows the model-view-template (MVT) architectural paradigm. It is intended to ease and accelerate the development of web applications by supplying a high-level, reusable, and modular set of components.

SQLite is a relational database management system (RDBMS) that is lightweight, serverless, self-contained, open-source, and embedded. It is created in the C programming language and is intended to be efficient, fast, and simple to integrate into diverse applications. SQLite, as opposed to typical client-server databases, is a file-based database engine that functions directly on disk files without the need for an external database server. We will use a different template and a different app to create our project.

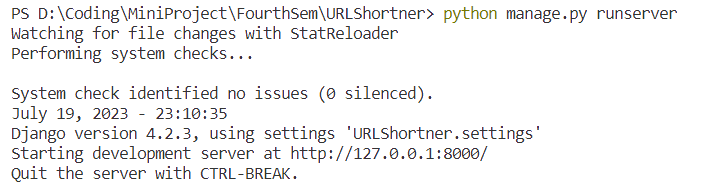
**3.1 Creating Project Space**

The command “python -m django startproject name\_project” is used to create a new Django project with the specified name (name\_project).

****

**3.2 Creating Server**

In Django projects, the command "python manage.py runserver" is used to start the development server. It starts a small web server that serves the Django application locally for testing and development.

****

**3.3 Creating URL routes**

The urls.py file in Django is critical for configuring URL routing and mapping for your web application. It allows the Django framework to determine how to handle diverse HTTP requests and create appropriate answers by mapping incoming URLs to specified view functions or classes.

When a user sends an HTTP request to your Django application, the Django URL dispatcher evaluates the URL and attempts to match it to the patterns established in the urls.py file. Django invokes the relevant view function or class to construct the response for that specific URL if a matching pattern is discovered.



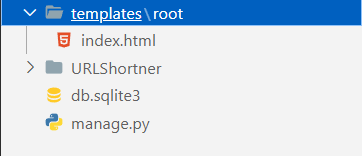
**3.4 Setting up new templates and Apps**

For this project, we will create our own templates and applications, and after that, we will update the information about the new template and app in the "setting.py" file.

The “settings.py” file in Django is a central configuration file for your web application. It includes parameters that define how your Django project behaves, such as database configuration and installed applications. This file is generated automatically when you start a new Django project.

**3.4.1 Creating New Template**

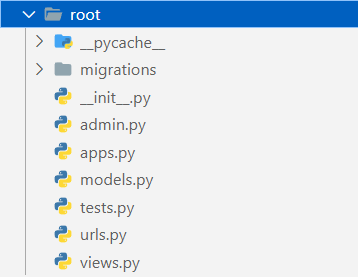
In Django, templates are a fundamental part of the framework's templating engine. Templates in Django are used to generate dynamic HTML (or other types of text-based content) that can be sent as responses to user requests.



And we created a “index.html” which contains the frontend of our project.

**3.4.2 Creating New App**

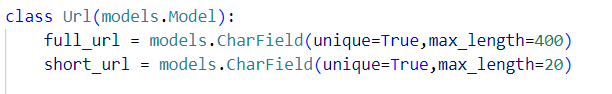
In Django, an app (short for application) is a modular component that serves a specific functionality within a Django project. An app is a self-contained package that contains models, views, templates, static files, and other components needed to implement a particular feature or functionality of a web application. Apps are designed to be reusable and can be used across different projects.



**3.5 Creating Database to store the URL**

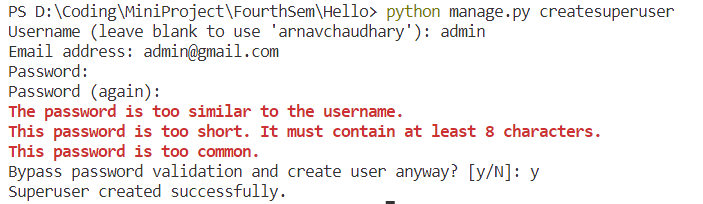
For creating the database in Django, we have a inbuilt virtual RDBMS called SQLite.

And for doing it we have to update the information in the “models.py” file.



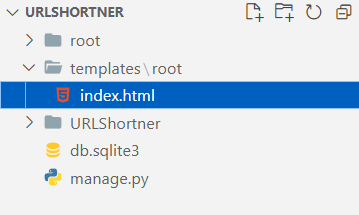
**3.6 Creating Admin to manage the database**

The management command "python manage.py createsuperuser" in Django is used to create a superuser account in the Django admin interface. Django's admin interface is a simple web-based interface for managing your application's data and performing administrative activities.

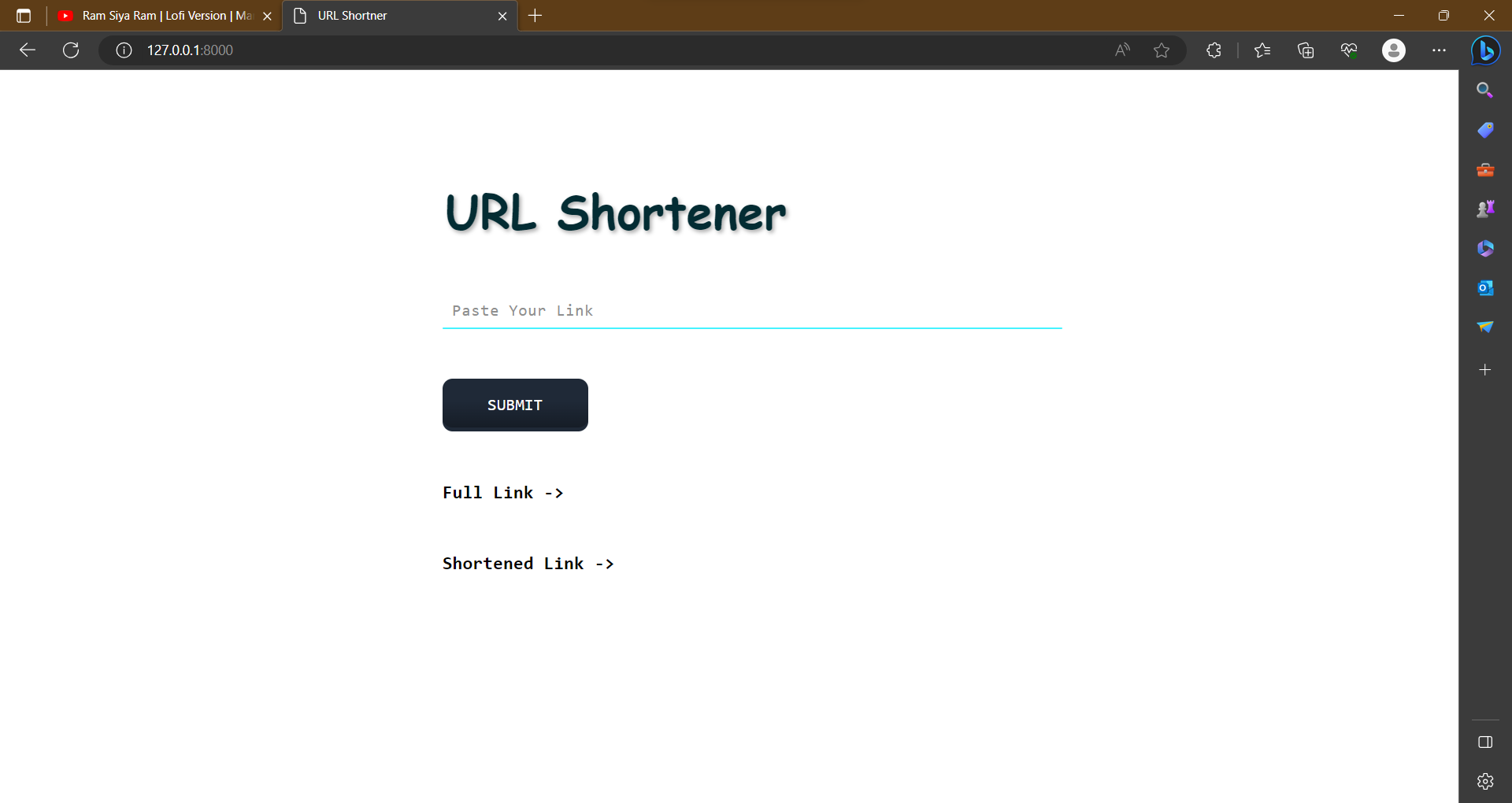


**3.7 Finally Creating the Frontend**

To create the frontend of our Project we have created a new Template and created the file “index.html”, containing all the HTML and CSS of our Project’s Frontend.



And finally, we have our project something like this,



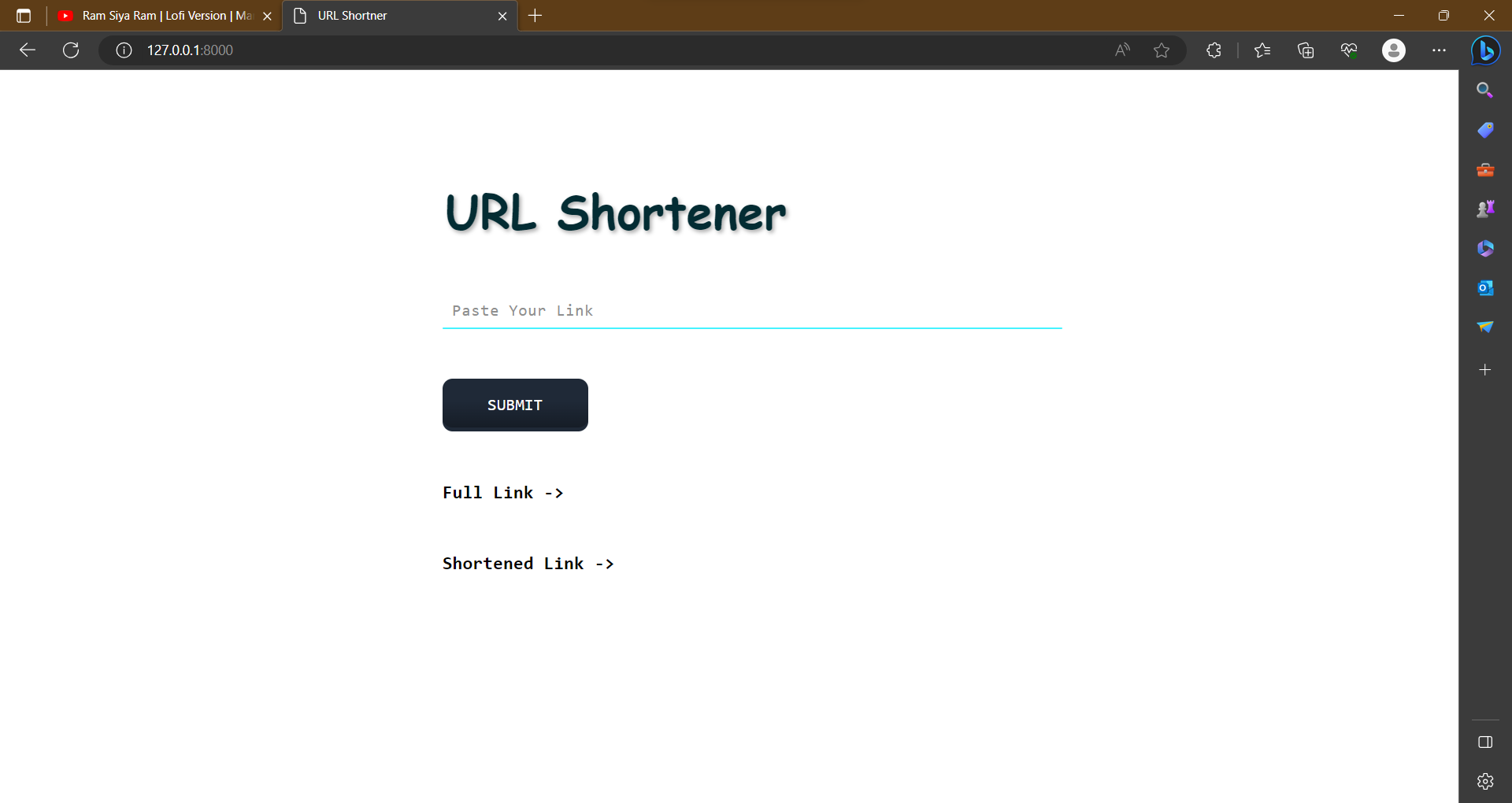
**Chapter 4**

**Result and Discussion**

**4.1 Result**

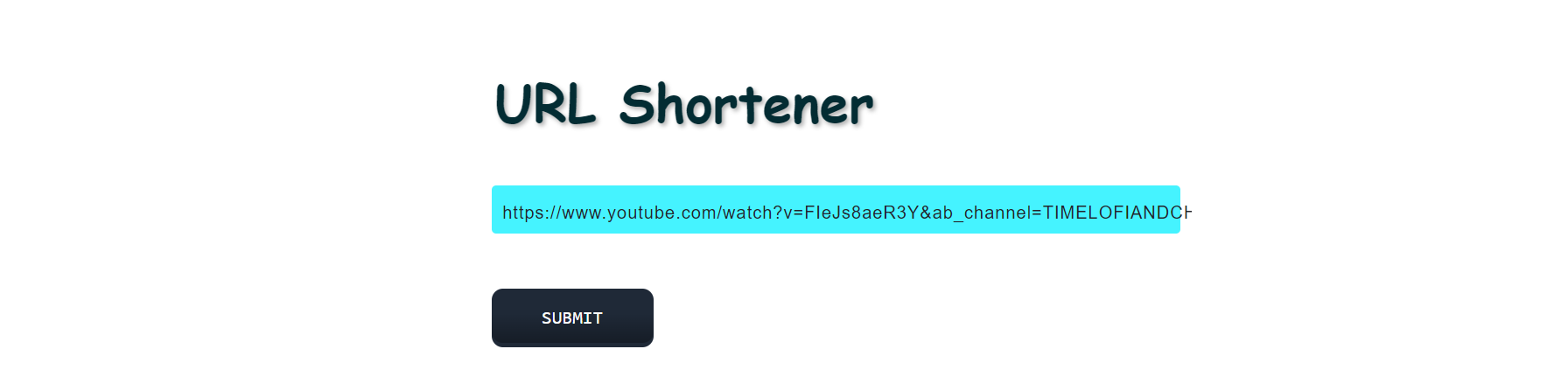
**4.1.1 Main Interface**

As we all know, a home screen is the first impression of any software project and application, related services. We have created the main interface using HTML and CSS and tried our best make it as minimalistic as possible and still able to give it the full functionality of a URL Shortener.

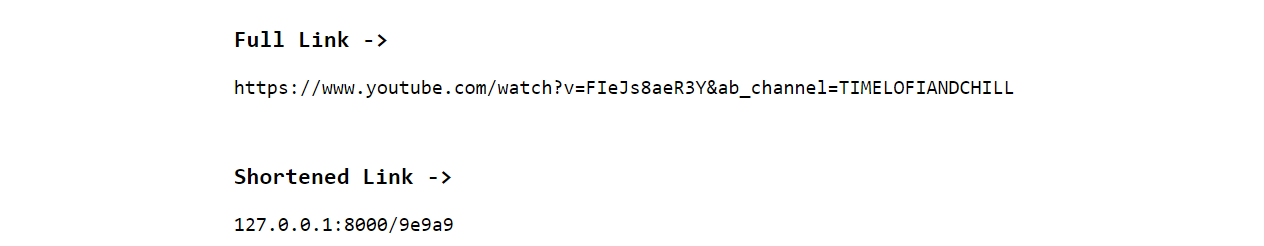


**4.1.2 How does it work**

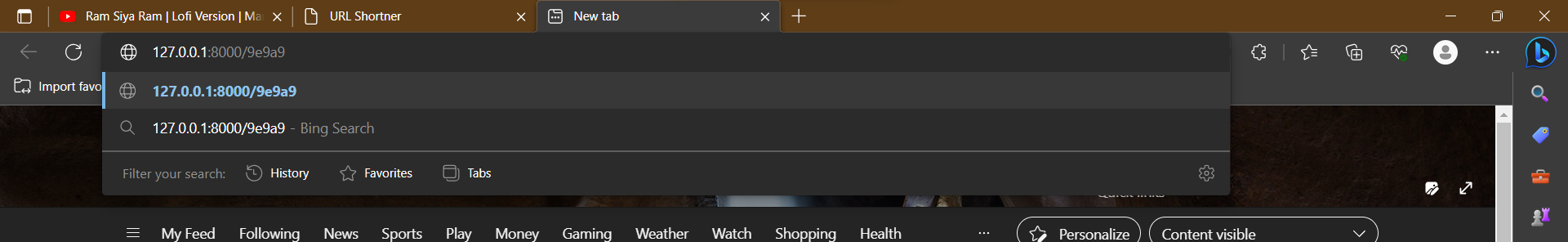
First copy the link that you want to shorten and paste it into the URL Shortener,



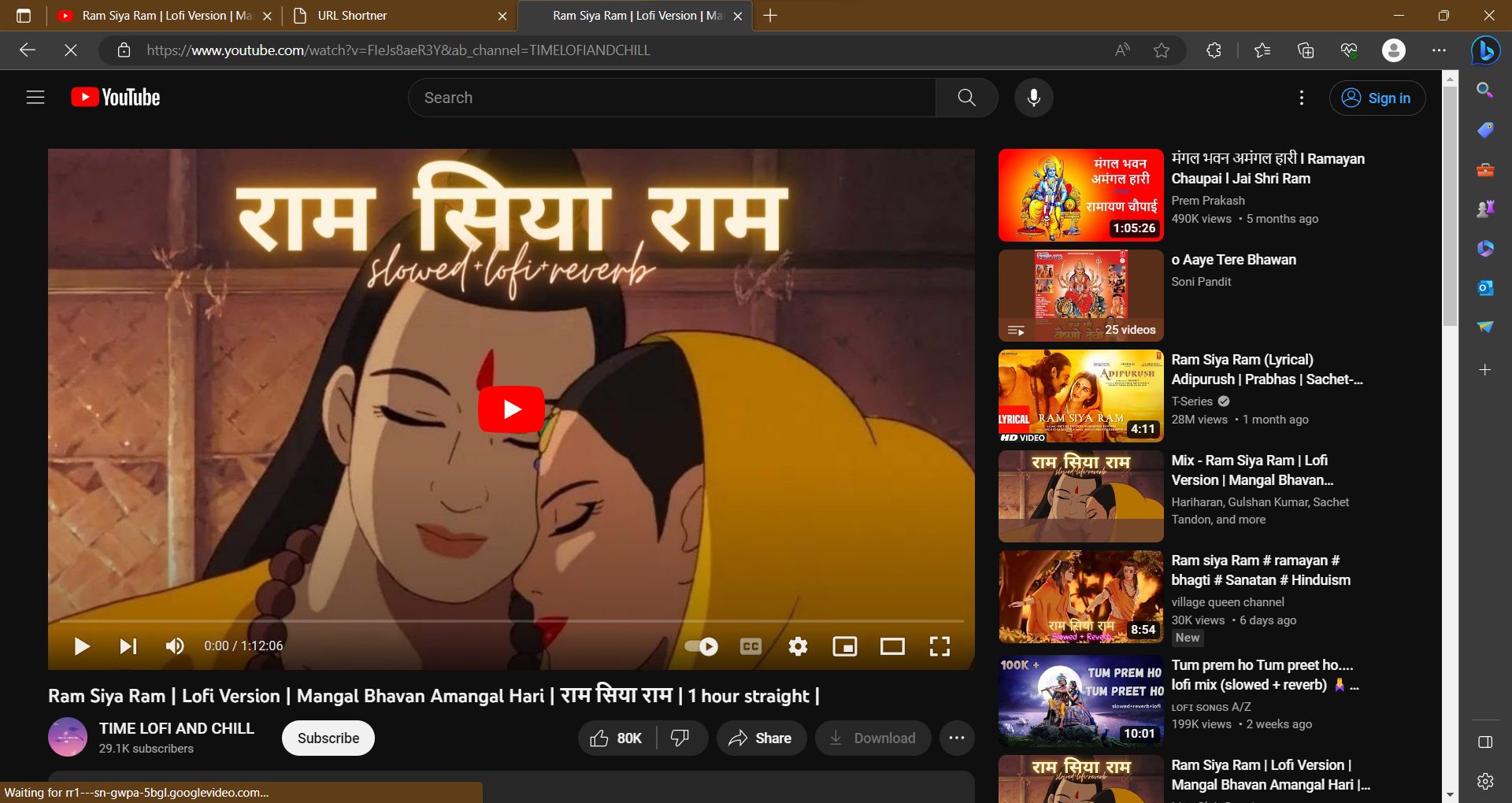
Then hit the SUBMIT button and after this you will get the Shortened link,



Now, copy the shortened link and paste it into your browser to be forwarded to the desired web page.



This will be redirected to,



**4.2 Discussion**

The use of Django to create the URL shortener proved to be a worthwhile venture, yielding various insights and outcomes worthy of further study. In this section, we will look at the important findings, discuss the difficulties encountered, and assess the overall effectiveness of the URL shortener app.

**4.2.1 Efficiency**

The robustness and versatility of Django were critical to the success of our URL shortener project. The framework's built-in functions, such as the URL dispatcher, ORM, and template engine, enabled quick development and clear separation of concerns. The Django REST framework also permitted the potential incorporation of APIs for future expansions, demonstrating the versatility of Django.

**4.2.2 Unique URL**

A vital part of our URL shortener solution was the selection of a unique key generation mechanism. The technique used efficiently reduced the probability of clashes, ensuring that each abbreviated URL remained distinct and functioning. Extensive testing and research confirmed the algorithm's dependability, making the URL shortener a trustworthy tool for maintaining and sharing links.

**4.2.3 Scalability**

In our pursuit of continual improvement, we have identified numerous potential future URL shortener advancements. Among the major goals are custom link aliases, link expiration options, and improved link classification. Furthermore, investigating the integration of social media sharing functions and third-party API support would give users with a more comprehensive experience.

**Chapter 5**

**Conclusion and Future Work**

**Conclusion:**

Finally, creating a URL shortener with Django has proven to be a fruitful endeavor. Long URLs are efficiently converted into shorter, more manageable connections, making it easier for consumers to share and access content. Django as the web framework enabled for speedy development while also ensuring a secure and sturdy system. We learned a lot about working with Django, establishing URL routing, and efficiently maintaining database models during this project.

**Future Work:**

While the URL shortener program is functional and achieves its core goals, there are several possible areas for future enhancement and growth. Some prospective projects include:

Analytics: Using analytics to track the number of clicks and user interactions with shortened links. This can provide useful information for analyzing user behavior and enhancing the application.

Custom Short URLs: Allowing users to build custom short URLs, allowing them to tailor links to their liking.

Expiration and deletion: Allowing users to establish expiration dates for short links or delete links they have made.

Using user authentication to provide customised dashboards for users to manage their shortened URLs and access analytics.

API Integration: Creating an API for the URL shortener application so that other developers and applications may smoothly integrate with it.

Link Categorization: Adding the ability to categorize links as well as filtering tools to help users better organize and manage their shortened URLs.

Enhanced Security: Constantly upgrading and enhancing security mechanisms to prevent misuse and maintain the integrity of shortened connections.

Mobile App: Creating a mobile app for the URL shortener in order to increase its accessibility and user base.

**References:**

[1] Django Documentation: Official Django documentation, available at https://docs.djangoproject.com/, was crucial for understanding Django's concepts and features.

[2] Django for Beginners: A comprehensive online resource by William S. Vincent, offering practical examples and explanations for Django development, available at https://djangoforbeginners.com/.

[3] Django Models Documentation: For details on creating models and managing the database, the official Django models documentation at https://docs.djangoproject.com/en/stable/topics/db/models/ was frequently referenced.

[4] Django URL Dispatcher: The official Django documentation on URL routing and views helped in setting up the URL shortening functionality, available at https://docs.djangoproject.com/en/stable/topics/http/urls/.

[5] Django Bootstrap: Utilizing Bootstrap for frontend development, the official documentation at https://getbootstrap.com/docs/ was consulted for creating a responsive and visually appealing user interface.

[6] Stack Overflow and GitHub: Community-driven platforms like Stack Overflow and GitHub repositories were valuable resources for troubleshooting and finding solutions to specific challenges encountered during development.

[7] Django Packages: Various Django packages and libraries were used to enhance the application's capabilities. References for these packages were obtained from the Django Package Index (PyPI) at https://pypi.org/.