B.TECH PROJECT REPORT

**U.R.L.S**

**Uniform Resource Locator Shortener**

Submitted By -

**Mr. Rahul Roy**

Registration Number – 112540110006

Roll Number - 25400111006

Project submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology**

in

**Computer Science and Engineering**

under the guidance of -

**Mr. Shibdas Dutta**

H.O.D. C.S. Department

**Kanksa Academy of Technology and Management**

**WEST BENGAL UNIVERSITY OF TECHNOLOGY**

May, 2015

**Kanksa Academy of Technology and Management**

PANAGARH-713148

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**HOD’S CERTIFICATE**

This is to certify that this Project **1-uP** is successfully completed by **Mr. Rahul Roy** under our guidance and supervision for the partial fulfillment of **Bachelor Of Technology in Computer Science and Engineering** under KANKSA ACADEMY OF TECHNOLOGY & MANAGEMENT of WEST BENGAL UNIVERSITY OF TECHNOLOGY.

**Mr. Shibdas Dutta**

H.O.D. C.S. Department, K.A.T.M

**Table of Contents**

[1. PROJECT DEFINATION Page 03](#ProjectDefinition)  
 1.1 Objective

1.2 Technical Breakdown

1.3 Deliverable

[2. PROJECT SCOPE Page 03](#ProjectScope)

[3. PROJECT PLAN Page 04](#ProjectReport)

3.1 Work Breakdown Schedule (WBS)

[4. PROJECT DETAILS REPORT Page 05](#ProjectReport)

4.1 Analysis

4.2 Architecture

4.3 Interface

[5. ADDITIONAL MATERIALS Page 15](#AdditionalMaterial)

**1.** **PROJECT DEFINATION**

**1.1 Objective**

Building a Uniform Resource Locator Shortening Service using a modern technology stack and cloud infrastructure.

**1.2 Technical Breakdown**

The project will consist of two chief components.

**Frontend:** A single page application which once loaded talks with the backend using a REST API

**Backend:** A simple API stack which will support the functionality of the frontend to create short urls, list them and display their analytics.

**1.3 Deliverable**

A single page application where a user should be able to **generate short links for long urls** (many short link per long url per user) and **see analytics** on a specific short URL by day, by hour, etc, and also click on the long URL aggregated from all short urls pointed to it. One long url may have many short urls. Analytics also include aggregation of **date/time of click, ip, location, user agent,** etc. Also include documentation explaining how to setup the application on a bare server.

**2. PROJECT SCOPE**

The project requires creating a RESTfull API driven Server to handle the **application logic**, **redirect traffic as necessary** and **log critical access data**, and **derive analytical statistics** and **business intelligence out of the raw access data**. The server needs to implement user profile based functionality and authentication services. Its primary and heavy workload is expected from traffic that requires redirect and logging of access data.

The project also requires the implementation of a Client Side Single Page Application served from the server which will handle view logic and communicate with the server to provide a user with a graphical user interface.

**3. PROJECT PLAN**

**3.1 Work Breakdown Schedule**

**1. Analysis time: 1 week**

* Technical Analysis
* Infrastructure Analysis
* Feasibility Analysis

**2. Architecture time: 2 week**

* Server Architecture
* Client Architecture
* Interaction Design

**3. Development time: 4 week**

* Functional Specification
* Technical Design Specification
* Pseudo Logic
* Code Implementation

**4. Testing and Demo time: 1 week**

* Integration Testing
* Negative Testing
* Demo

**4. PROJECT DETAILS REPORT**

**ANALYSIS**

**Technical Analysis**

Technology Stack to be used is Node.Js for Application Server and MongoDB for database. MongoDB is optimized for and provides excellent support for high IO. Node.Js is a very resource friendly Web Server Platform which provides exceptional load capacity with minimal hardware requirements.

**Infrastructure Analysis**

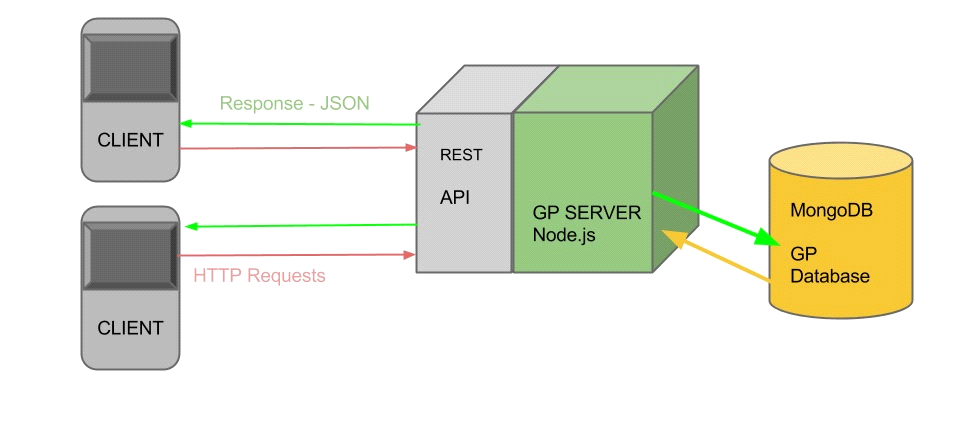
OpenShift is a reliable and trusted cloud platform provided by Red Hat for hosting the application server and database server. Having both in the same cloud infrastructure ensures less network latency for Application - Database data CRUD operations.

Client Application Prototype Platform is Windows Phone OS version 8 because it provides native maps support and is superior in this regard to the competition due to shared map oriented resources which exposes native platform APIs.

**ARCHITECTURE**

**Server:**

High Level Architecture



**OPENSHIFT CLOUD PLATFORM by RED HAT**

To support scalability, the Database and Application server is running on different Gears. This allows the Application Server to spawn additional instances during heavy traffic without affecting data availability. This prevents the use of an in-memory key value map dependency. The Application is designed to support this process.

**Application Host Configuration:** 1 Small Gear with **Node.js** 0.10 cartridge.

**Database Host Configuration:** 1 Small Gear with **MongoDB** 2.4 cartridge.

A gear is a resource-constrained container that runs one or more user-specified software stacks, also known as cartridges. Each gear has a limited amount of RAM and disk space. If an application needs more resources, it can be configured to automatically use more gears.

**Small Gear Specifications:** Memory 512 MB

Storage 1GB

**Authorization**

Every Client is assigned an Authentication Key hereby referred to as access key. This Access Key is Unique and Linked to only one client instance.   
  
On first launch of the Client Application, an Access Key is requested against the ANID2 property of the Registered Device User. Once assigned, the Access Key is stored on the device for subsequent access. If any requests to the server results in an Unauthorized Access Message, the Client will request the key again. This system is plugin friendly and will allow introduction of other security and attack detection features without interfering with functionality.

**Communication Protocol**

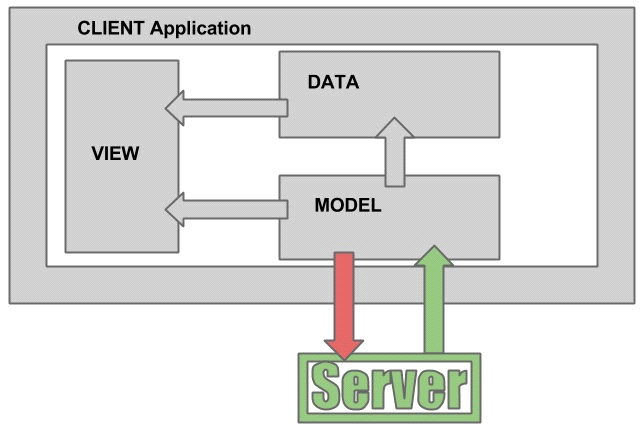
The server exposes a **REST API** with which the Client Application can interact with the server following a static handshake protocol defined in the application design.

The RESTfull architecture is implemented using stateless Hypertext Transfer Protocol without any session implementation. **HTTP** functions as a request-response protocol in the client-server computing model. Security is serviced by Key Based Authentication.

For further details refer to [Annexure 2](GP%20Project%20Documents/GP%20Software%20Design%20Description%20(SDD).docx) – GP Software Design Description (SDD) and [Annexure 3](GP%20System%20Architecture%20Documents/GP%20Server%20Architecture.docx) - Server Architecture Document.

**Client:**

High Level Architecture

****

**View Component:** This handles the rendering of UI Elements and presenting the data in a user friendly layout. Comprises of Map Elements, Map Push Pins, Buttons and other layout elements. The View Layer reads from the Data Layer. View Layer updates and re-renders are triggered by the Model Component.

**Data Component:** Comprises of all the JSON Serialization and Deserialization Helper classes and Data Structures which represent the in memory data from the server. The Data Layer can be read by the View Layer and modified by the Model Component.

**Model Component:** This handles all the interaction between the client and server. This component alone has server and data access. It is responsible for triggering view updates and fetching real-time data from server. It also handles user authentication and location broadcast.

**Interaction Architecture:** API Endpoints Definition

**[GET] /**

- Ping server for status.

**[GET] /:shorturl/**

- Requests to this path are redirected with a status of 301 to the long urls. The access headers and IP of the request is stored for processing.

**[GET] /api/v1/domains/**

- Returns a json encoded array of supported domain options as set in the config.js file on server.

**[POST] /api/v1/login/**

**-** Intended to authenticate and verify existence of user.

*Variables: username, password.*

**[POST] /api/v1/signup/**

- Used to create new user profile on server.

*Variables: username, password.*

**[POST] /api/v1/lurl/new/**

- Used by Client to push destination of user to server.

*Variables: username, password.*

**[POST] /api/v1/lurl/get/**

- Used by Client to fetch list of nearby GP users.

*Variables: username, password.*

**[POST] /api/v1/lurl/addsurl/**

- Used by Client to get list of users who have a destination close to the one specified.

*Variables: username, password.*

**5. ADDITIONAL MATERIALS**

[Live Demo -](http://urls.rahulroy9202.in/)

URL: http://urls.rahulroy9202.in

Host: OpenShift Red Hat Cloud Platform.

Author: Rahul Roy

[Project Source Code](https://github.com/rahulroy9202/urls/) -

URL: https://github.com/rahulroy9202/urls/

Host: GitHub

Author: Rahul Roy

Project Documentation -

URL: https://github.com/rahulroy9202/urls/

Host: GitHub

Author: Rahul Roy