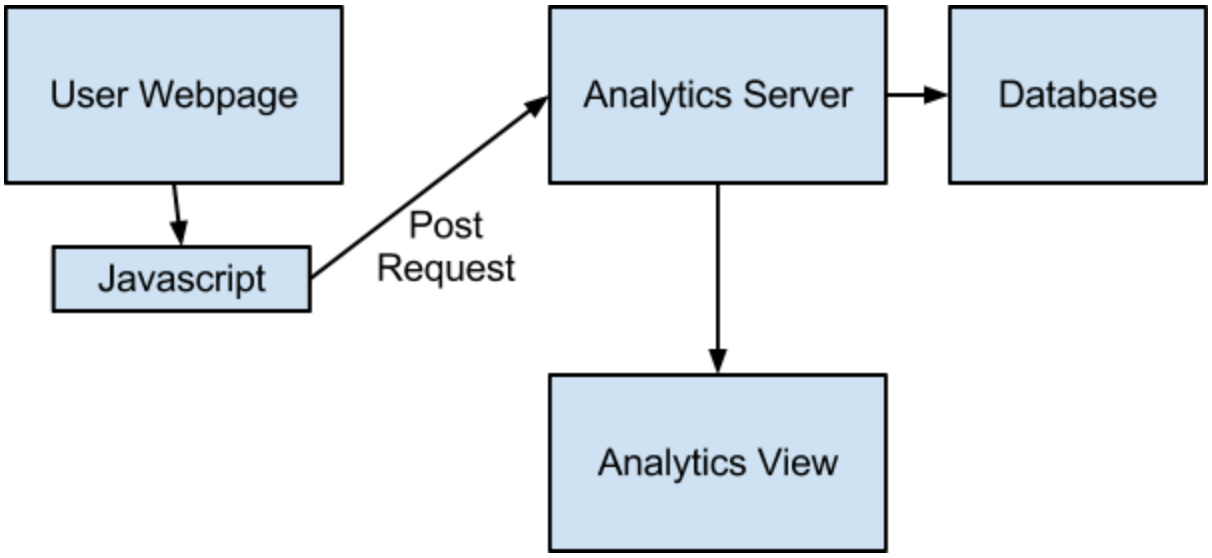
Overview:

Purpose and goals: This system is a simple web analytics platform. It tracks the visits to individual pages by means of an embedded javascript that the user pastes into the code for each page of their site.

Context diagram:



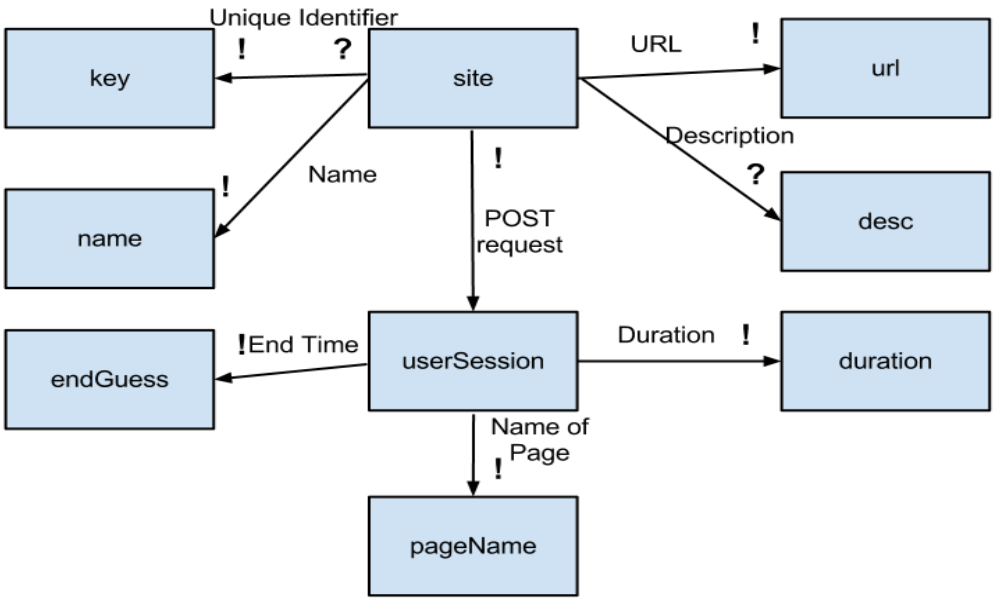
Concepts:

Key concepts:

The key concepts in this project are allowing a webpage to interact with a server via http requests in javascript. A user embeds a piece of javascript code in their site. This code makes POST http requests to a server. The server tracks the page views and duration for the given website. This information can then be displayed to the user.

Object Model:

There are two main structures in the database: sites and userSessions. Each site represents a website that is being tracked - including a unique key, a url, a desc, and a name. Each userSession represents a user visit to a page and contains the endGuess, the duration and the pageName. A new site is created when a user submits a form and a userSession is created when the server receives a POST request to a particular site.

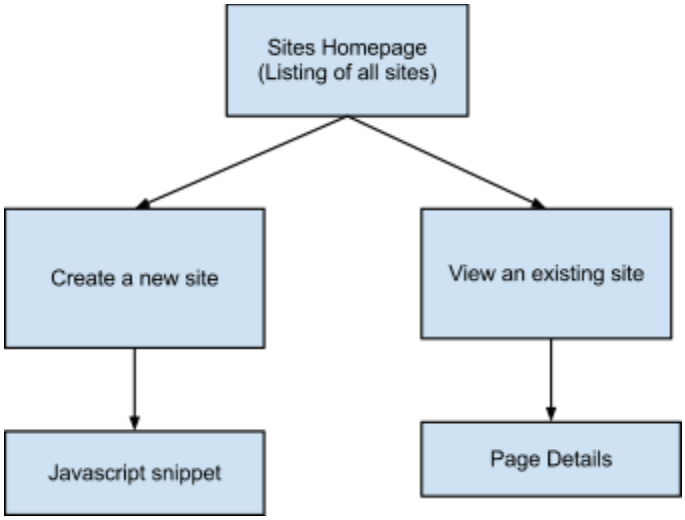


Behavior:

Feature descriptions: The main features are the ability to track the time users spend on each page on a particular site and the total number of page views for that site. This information is then displayed to a user.

Security concerns: There are several security concerns including privacy of information - currently anyone can view the web analytics data for any page tracked. In addition, passwords should be salted and hashed before being sent and the domain which the request comes from should be checked against the site that is being tracked to prevent malicious requests that are not actual page views (for example just running javascript snippets). These are just a few of the possible security concerns. However, we were not required to implement a security system in this project, hence the large number of security gaps and a lack of basic security features.

User interface: The user interface is extremely simple. A user initially sees a page listing all the currently tracked websites. The user can then view the details for any one site or create a new site. In addition they can edit and delete old sites. When a user clicks to view the details of a site, they see the average visit duration for the site overall as well as individual pages and they see the total number of page views.



Challenges:

Design Challenges: There were several design challenges - the first was cross domain scripting - enabling a webpage to make a call to the server. This was resolved by allowing any domain to access the server through setting the cors headers.

The second was tracking the time a user spent on a given page. This could be done in several ways. First you could track a user through cookies and watch their progress through pages - as they load each page you can tell the time they spent on the previous page. This is server-side tracking. While this would allow a better tracking of users as they move through the site and be more platform independent, it is also much more complicated to implement and increases server load. The solution I choose was client side tracking - have the javascript track the time the user spent on the page and send this as the duration once in the unonload request.

Then there was the database model. There are numerous options available - including tracking users, pages, sites, etc. However I chose to focus on a two-layer model. The first layer is the sites that are being tracked. Each site is then associated with a number of userSessions which record each page view. While this has the disadvantage of taking longer to compute the average time spent on pages, it does not need to worry about synchronizing the average - which would happen if you were updating them as page views were posted.

Evaluation:

Critique:

From the user's perspective, the application is fairly simple and an easy text-box allows copy and paste of the web analytics code. The problem however is that the user interface is numbers-based and that the display of pages can become long and difficult to parse. In addition there is a notably lack of buttons and graphs and images. I was attempting to work with twitter bootstrap but did not get it to work successfully. Overall the application works well and performs

as expected. Besides obvious security flaws, the least successful part of this project was the user interface. The current server collects more information about the visits (such as endtime) than I had time to display. This should not have been collected if it would not contribute to the display for the user. Overall however, the code is simple and relatively minimal. The first thing to improve would be the user interface followed by security features.

Reflection:
The most successful aspect of the project was learning to use the scaffolds created by ruby and to simplify design modules to minimize code and make it more elegant. The least successful was learning how to add a user interface and also the attempt to push to heroku - I ended up copying the app several times which simplified by database structure and code, but used up development time. Next time, I have a head start on understanding twitter bootstrap for a user interface and have learned how database migrations work.