CS455A: Introduction to Software Engineering

DataCloud Analytics

Analytics as a Service

Akash Waghela, Pranshu Gupta

Introduction to Software Engineering [CS455A] – Project Report
Indian Institute of Technology Kanpur
Department of Computer Science & Engineering

Contents

Introduction....3
User Interface....3
The User Registration Screen....3
The User Login Screen....3
The Dashboard....4
Design Pattern....5
Architecture....6
Use Case View....6
Logical View....7
Process View....8
Development Decisions....9
Why Django?....9
Why D3.js?....9

Introduction

The software we have implemented is a web service that will be used by businesses to visualize their consumer and product usage data to inform and measure their marketing. It is an instance of Analytics as a Service.

Analytics as a Service "AaaS" refers to the provision of analytics software and operations through Web-delivered technologies.

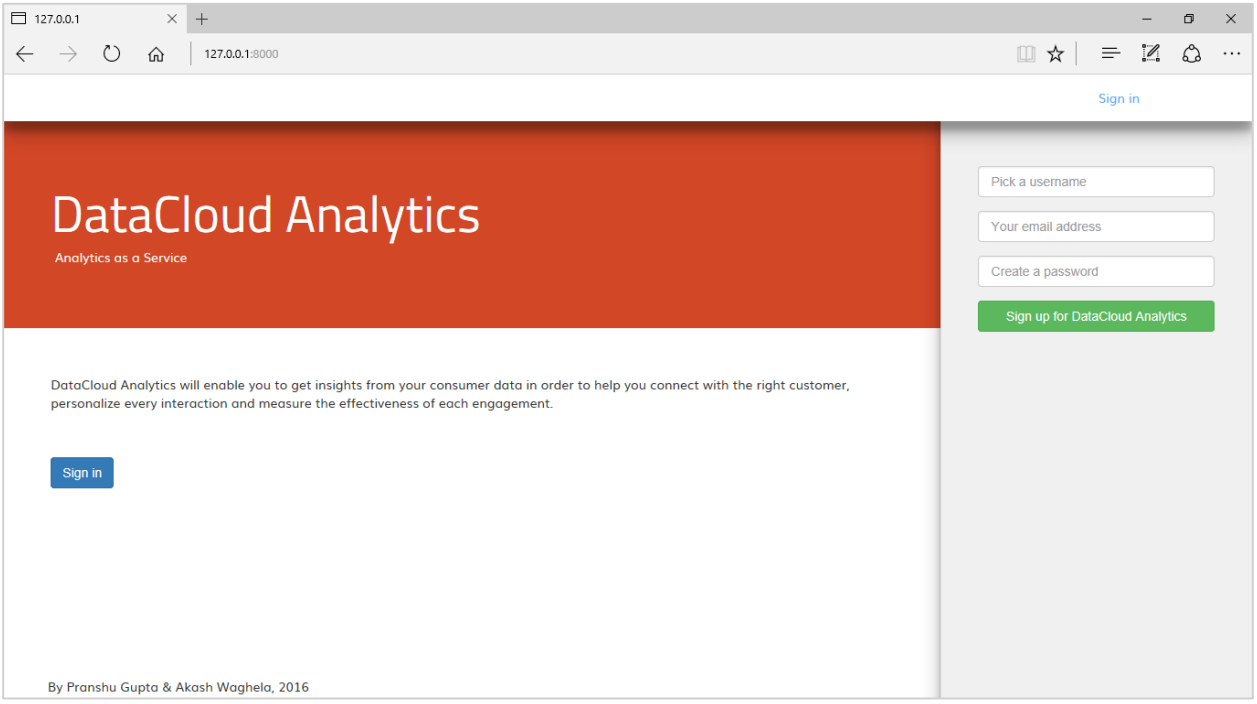
User Interface

The user interface of our application consists of the following screens:

1. The User Registration Screen
2. The User Login Screen
3. The Dashboard
    a. General Analytics Screen
    b. Geographical Activity Heat Map Screen

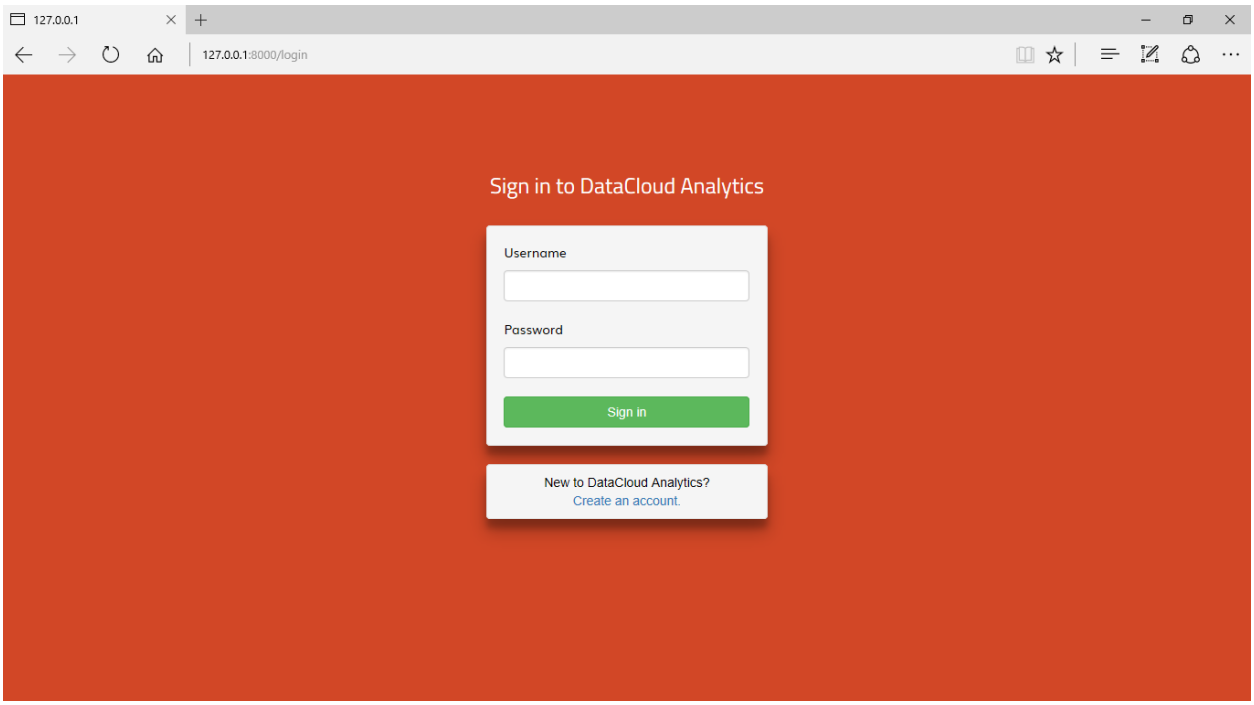
The User Registration Screen

This is the landing page of our application. It provides some general information about the service along with a user registration form.



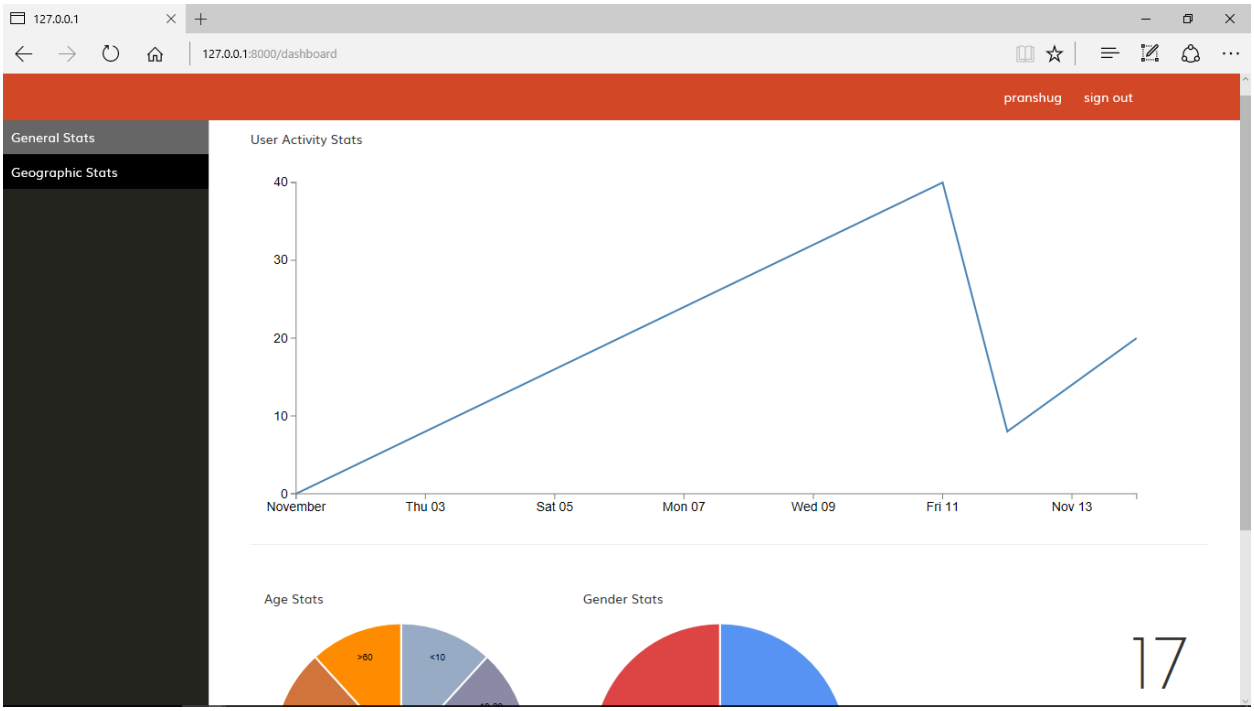
The User Login Screen

This is the screen where the user can log in to our application and utilize the services provided. If a new user directly lands on this page, there is an option to create a new account here too.

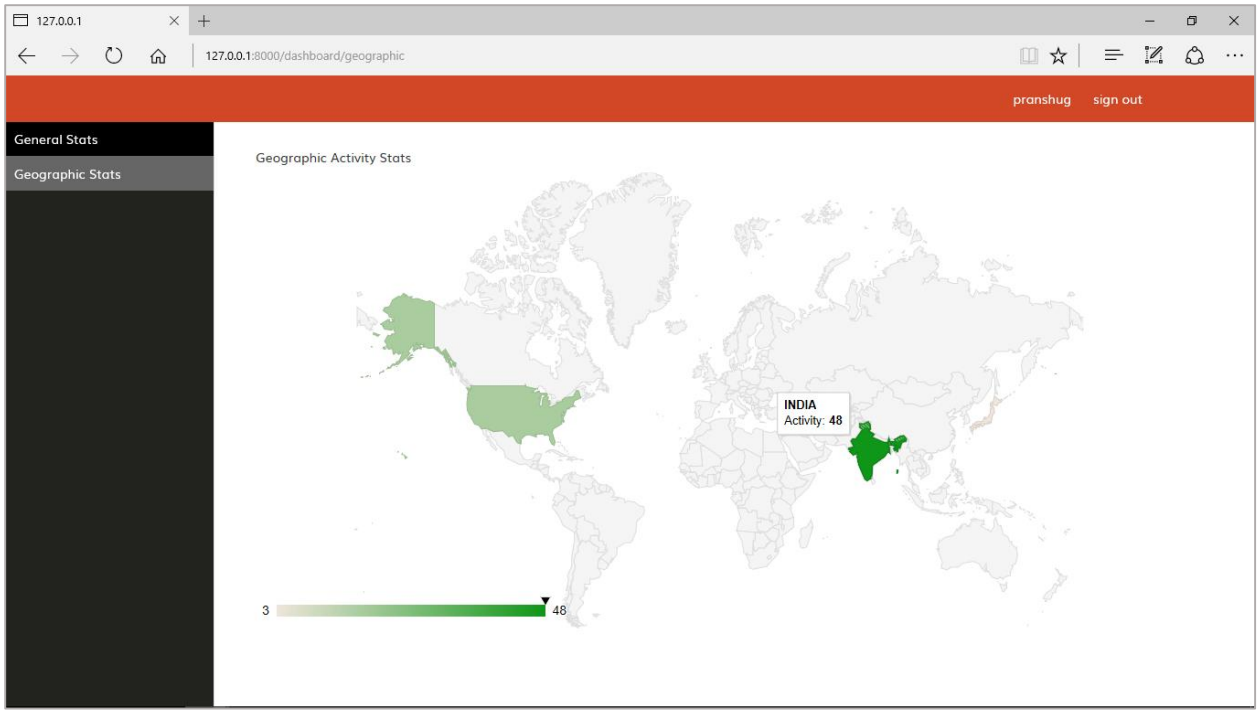


The Dashboard

The dashboard is what the user sees after login. The layout of this page has three parts, the top header, the navigation pane and the content view. The navigation pane provides the user with links that can be clicked to view different categories of analytics visualizations.



Currently, we have provided two such categories: General and Geographical Activity Stats.



Design Pattern

We have followed the design pattern defined by Django Web Framework for developing our application. Django is said to follow MVC, although some say it is MTV. MTV (Model Template View) is very similar to MVC (Model View Control) but there is a subtle difference.

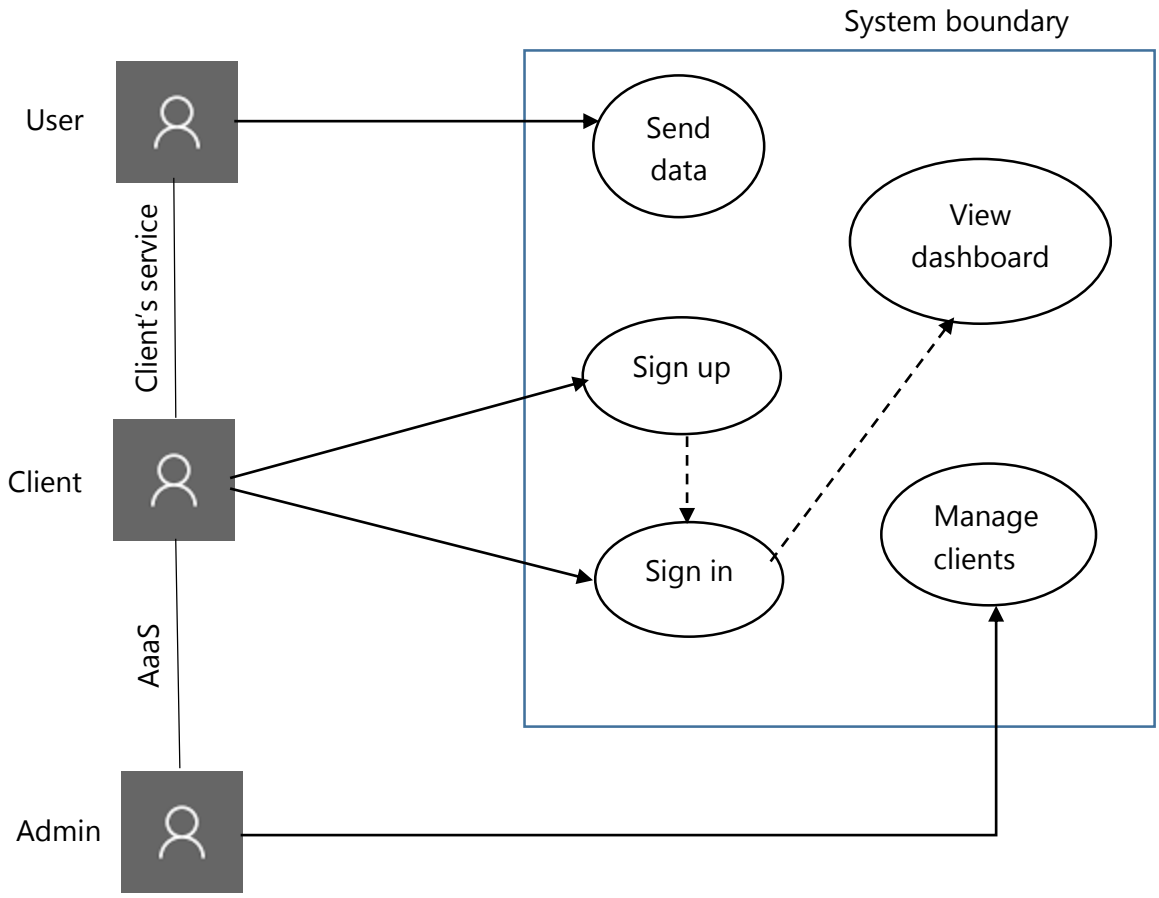
In MTV, a request to a URL is dispatched to a View. This View calls into the Model, performs manipulations and prepares data for output. The data is passed to a Template that is rendered and emitted as a response. Ideally, the controller is hidden from view.

In MTV the template describes the data that gets presented to the user, not necessarily how it is presented, but what is presented. In contrast, MVC suggests that it is the controller's job to decide how the data is presented while the view is strictly responsible for how the data looks. So, a 'template' in MTV loosely replaces a 'view' in MVC. While the job of an 'MVC Controller' is handled partly by a 'MTV View' and partly by the framework itself.

Architecture

Use Case View

We have three actors in our application, the *Service Provider* (admin), our *Client* and the *Users* of the client's service for which the client has opted for our "AaaS" offering. The following diagram shows how these three actors and our system interact.



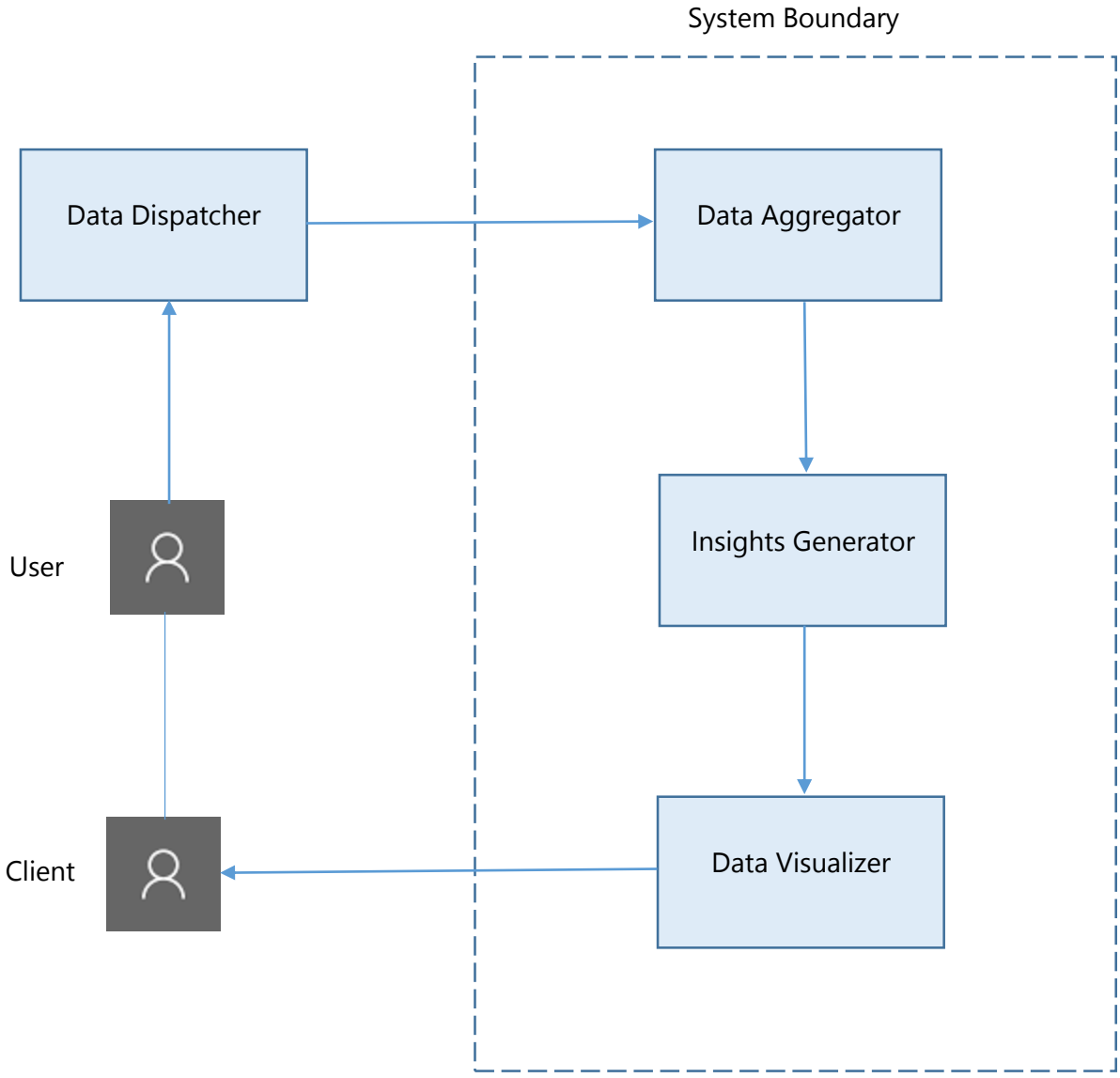
1. Sign UP/ Sign In – The actors triggering this use case are the clients (business users) who have opted for our "AaaS" offering.
2. View Dashboard – This use case enables the clients to visualize data related to the usage of their own services which send data to our system for analysis.
3. Send Data – The actors triggering this use case are the users who consume the services that are provided by our client. This is done by their browser with the help of a script included in the client's service as per our API.
4. Manage Clients – This use case is triggered by the service provider of our system, where client accounts can be managed.

Logical View

Following are the different modules of our system

1. Data Dispatcher – A script that collects data from the user of the client's service and sends it to our system
2. Data Aggregator – The module responsible for collecting the received data and populating the database.
3. Insights Generator – The module responsible for generating insights that will be used to create visualization on the dashboard for the client.
4. Data Visualizer – The module responsible for creating appropriate visualizations for the insights generated by the Insights Generator.

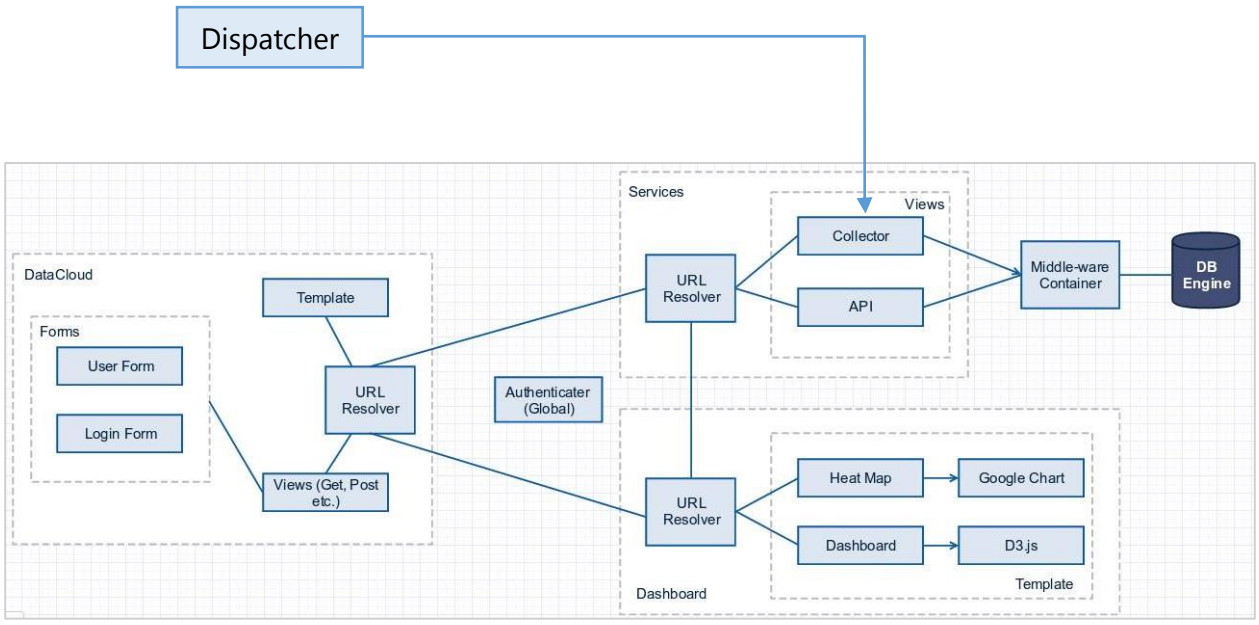
Following diagram describes the logical view:



Process View

The entire application can be divided into the following components:

1. Authenticator/Authorizer – Globally accessible, used for authentication while writing and reading data.
2. URL Resolver – Every Component has this block to redirect http requests to appropriate end point.
3. Data Collector – This process collects data sent by the data dispatcher.
4. Service API – This process is activated by the web-application while querying Database for various entries based filtered on the basis of a given parameter. This is the insights generator at run time.
5. Forms – Provides UI based functional requirements required by Login/Sign Up.
6. Views – Binding data to the templates and sending http responses.
7. Templates – Define layout of the pages visible to the client.
8. Dashboard – The run time component for the data visualizer. It has two parts the Geographical Heat Map that uses Google Charts and a general data visualizer which uses D3.js



Development Decisions

Why Django?

We chose Django because of following points:

1. Python is a really nice, elegant & powered language. There are guidelines how to write and format your code and there is, well most of the time, a clean structure in your code, no matter what you do. Python follows the principle "code is much more read then written".
2. The Django ORM is an incredibly powerful database tool. It handles creation of your database, as well as insert, update, delete queries and some quite advanced querying - although it's not perfect. It supports multiple databases - MySQL, PostgreSQL, Oracle & SQLite are all supported out-of-the-box assuming you have the relevant Python libraries installed. You write your database as Python class and query it using Python. You do not have to write one line SQL by yourself.
    a. *We have tested our application for both SQLite3 and PostgreSQL without any code modifications*.
3. Forms are not the most fun thing. While Django doesn't make them fun, it at least does a lot for you. You define some fields and how you want the basic validation to work, and Django creates the HTML adds the error messages and cleans the data so you don't get anything unexpected. The Django forms framework can even generate and update your database from a database model you create, make your job even easier.

Why D3.js?

We chose D3.js because of the following points:

1. Using D3.js we separate the data analysis and data visualization components of our web service. Data Analysis is done on the server side with Python and the results are sent to the client side in JSON format which can be used by D3.js library to make visualization independently and without putting the load of graphics rendering on the server.
2. In fact Shiny itself is built upon D3.js and Google Charts. And D3.js has huge number of examples available on the web to get inspired from.
3. We have also used Google Charts. We found that the code we have to write is less verbose as compared to D3 and in a few cases quality of charts rendered by Google Charts is superior to D3 (e.g. Geographical Charts). We might consider moving entirely to Google Charts.