CS 457 Fall 2016

Assignment 1

Team 9:

Naga Mounika Dandamudi - 9

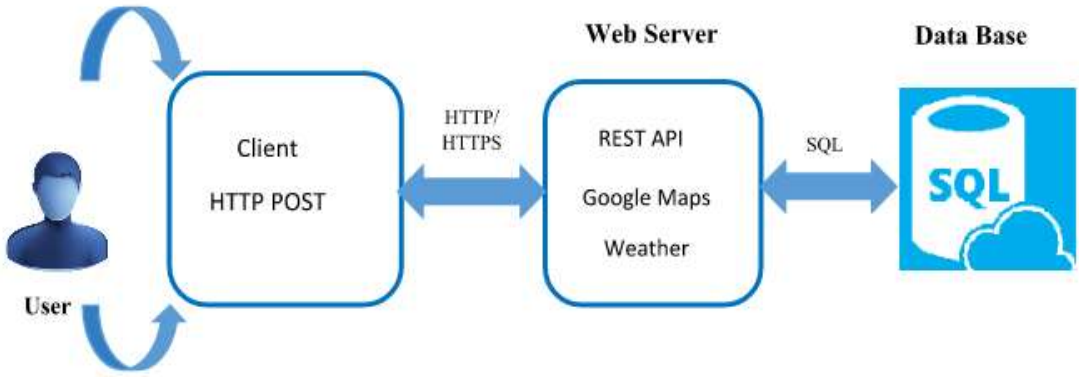
Bhulakshmi Makkena - 21

Bhargav Krishna Velagapudi – 39

Rajendar Prasad Palagarla - 29

1. Come up with an architecture diagram for the use case. Be creative and use the slides on sample architectures to get a peek into how architectures can look like. But most of the work would happen through team discussion and lot of googling. Document the factors that led to the architecture diagram and also explain in brief the role of each component.

   The use case that we choose is Nomad
   Nomad is an application which gives best route from source to destination along with the weather information at particular times. It also has to store the history of past journeys and suggestions to visit around the area.
   In order to implement our use case, we are using **Web mashup architecture**,



   There are three layers that mashups are composed of:
   1) **Presentation or User Interaction Layer**: This is the user interface of the mashup where technologies such as HTML, CSS, AJAX, and XML are used to make the mashup as visually appealing and functionally efficient as possible.

2) **Web Services**: Used to connect to the products functionality with API services. Tools such as XMLHTTPRequest and SOAP are used to do this.
3) **Data**: How the sending, storing and receiving of data is done often XML is used.

The user interaction layer interacts with the web server and gives the response usually in JASON format. The web services layer has different REST services.

For our use case, we are using Weather API and Google map API to achieve the functionalities. The front end will be written in HTML and Angular JS.

Organization within mashup architecture is essential in ensuring that it will deliver the necessary functionalities quickly and efficiently and using the appropriate protocols and design principles will enable mashups to be designed quicker.

2. Choose an architecture style suitable for the use case you have chosen. Document your explanation on why you picked that style.

Brittleness – Due to Screen-Scraping – In the long term Web APIs change format (without telling the Mashup)

• **Loosely Coupled?** – The mashup will not run if any of the backend API providers are down

A mashup is a new service that exhibits the combined functionality or content from different existing sources such as web services (through use of API's). A web mashup architecture mainly involves aggregation, combination and visualization and makes the existing data more clear and useful.

A mashup architecture gives us answers faster,

A mashup architecture improves the resource use.

Mashups typically works at UI and API levels.

It is also possible to bypass the application and directly access its database (Data Integration).

Mashups use these lightweight Web protocols to access and compose together remote software components and data sources.

Mashup Properties:

• Aggregate content from more than one source – Public Web Service APIs – Screen Scraping (scrAPIs) of existing Web sites – Additional local data owned by Mashup provider – User-provided information (shared wiki or "private")

• Lightweight programming effort – HTML with some JavaScript is enough to get started

• Interactive Web application – Can also provide its own API

• Ad-hoc composition – Not concerned with long term stability and robustness

Mashup Pros and Cons:

Pros:

• An advantage to using mashups is that it allows for the reuse of existing applications. So, instead of spending time developing a component for your website, you can instead use a preexisting one to implement the feature that you want on your website.

• Since you are saving time not having to implement certain features, you are able to more rapidly develop applications. This allows you to get through several more prototypes in a shorter amount of time then if you were to develop all the components yourself.

• Mashups are designed so that most of the work is abstracted behind the scenes so that the user simply only needs to learn how to implement it. This allows users who normally could not implement such features on their own, to be able to incorporate these features.

• The cost of application development is reduced greatly. Because there are all these preexisting applications which are easy to develop, money will not need to be spent on long development schedules as well as the training required to teach developers to implement the features they wanted from scratch.

Corns:

• User has no control over the quality or the features in the application component they are implementing. So the user is dependent on the other developers to make sure that the application does what it intends to and has no bugs.

• This dependence on the other developers also does not guarantee that this application component will be continuously supported. So if the service or API gets discontinued, the user will have no choice but to replace it.

• Scalability: There is no guarantee that the service you are implementing will be able to accommodate the traffic your website will get if it grows bigger.

• The contents used in the mashup service are not guaranteed to be secure. So if you are a large enterprise or have sensitive data, incorporating the mashup API may pose a security concern.

• There are no standards for the development or application of a mashup, so this poses yet another difficulty in designing and implementing security mechanisms to ensure the integrity of the data being used.