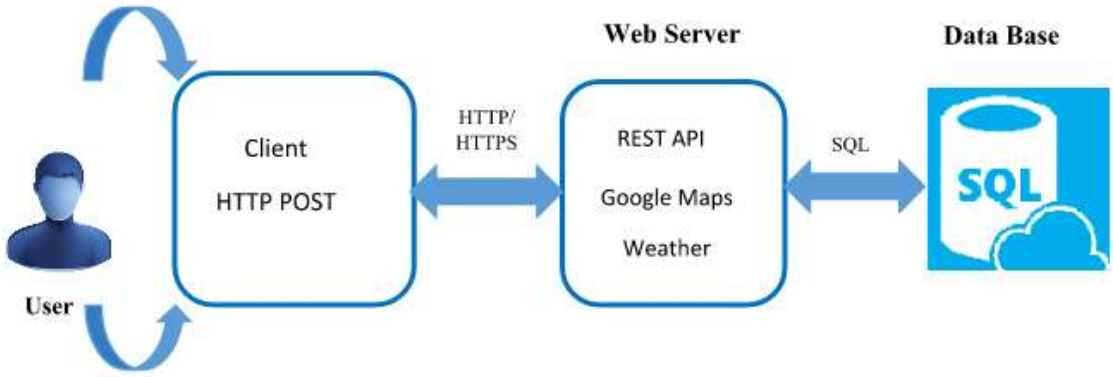
# CS 457 Fall 2016

# Assignment 1

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 usually operate UI and API levels.

With mashups it is possible that we can sidestep the application and can directly access its database.

Mashups utilize various lightweight web protocols to access to and combine together different software components and data sources.

## Mashup properties

Aggregates content from more than one source- these sources can be web services such as API's

lightweight programming – HTML with some javascript is sufficient to begin with

interactive web application – it can also provide it's own API

Adhoc composition – it does not concern about the future use and robustness.

## Mashup pros and cons

**Pros:**

- Advantage of utilizing mashups is that it takes into consideration the reuse of existing applications. Thus, rather than investing energy building up a segment for your site, you can rather utilize a previous one to execute the element that you need on your site.

- Since you are sparing time not implementing certain elements, you can develop applications more quickly. This permits you to traverse a few more models in a shorter measure of time then if you somehow managed to build up every one of the segments yourself.

- Mashups are planned so that the vast majority of the work is behind the scenes, so that the client just needs to figure out how to execute it. This permits clients who ordinarily couldn't actualize such features all alone, are able to incorporate these features.

- the expense of developing the application is lessened enormously. Since there are all these prior applications which are anything but difficult to create, there is no need of investment on long improvement plans and additionally the preparation required to instruct designers to implement the features they are willing to have from the scratch.

**Cons:**

- Client has no influence over the quality or the elements in the application part they are implementing. So the client depends on alternate designers to ensure that the application does what it plans to and has no bugs.

- This reliance on alternate designers additionally does not ensure that this application part will be continuously supported. So if the administration or API gets ceased, the client will have no real option except to replace it.

- Scalability: There is no guarantee that the administration you are executing will have the capacity to accommodate the traffic your site will get in the event that it becomes greater.

- The contents utilized as a part of the mashup administration are not ensured to be secure. So in the event that you are representing a large enterprise or have a delicate information, incorporating the mashup API may result in a security concern.

- There are no models for the advancement or utilization of a mashup, so this results yet another trouble in designing and implementing security systems to guarantee the trustworthiness of the information being utilized.